CS Exit Ethiopia

Questions & Worked Answers

Computer Science Exit Examination

294 complete questions and 4 source variants

1 selected course | 18 November 2026

Selected courses

01. Object Oriented Programming

https://csexitethiopia.com

abelasnake2003@gmail.com | Telegram: @Abechu7

Course contents

Each course begins on a new page. PDF bookmarks provide question, study-note section and revision-topic navigation.

Object Oriented Programming 3

Reading the answers

Every question includes its answer, explanation, choice-by-choice contrasts and available evidence. Primary note proof quotes the study notes. Related reading and module passages are labelled separately; a module reference is not presented as primary note proof.

Object Oriented Programming

Questions & Worked Answers

294 questions + 4 source variants

Question 1 | Q00213

When overloading a function, what must be true?

A. The names should be different with different number and/or types of parameters.

B. None of the choices are correct

C. The names should be different with the same number and/or types of parameters.

D. The names should be the same with different number and/or types of parameters.

E. The names should be the same with the same number and/or types of parameters.

Hint

Compare names and parameter signatures before considering return types or access modifiers.

Answer D: The names should be the same with different number and/or types of parameters.

Overloads retain the method name and change parameter count or parameter types; return type alone cannot distinguish them.

Why each choice is right or wrong

A - Incorrect: Different names do not overload one function

B - Incorrect: D satisfies the overloading rule, so the statement that none of the choices is correct is false.

C - Incorrect: Different names define separate methods

D - Correct: Same name plus different parameter list meets the definition

E - Incorrect: Same name and same parameter signature cannot distinguish overloads

Primary note proof

Study-note section OOP12

> Overloaded methods share a name but have different parameter types, counts or type order. Return type alone is insufficient in Java and C++. Overloads can also arise through inherited methods; “same class only” is a teaching simplification. f(int), f(double) and f(int,int) are distinct, but changing a parameter name does not create an overload.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 11 | lines 7-18

Method Overloading
• Is a process of having the same name for different methods as long as their parameter list or data types in the parameter list is different?
• The return type alone is insufficient to distinguish two versions of a method.
• Parameters are key identifiers of OM.
• Is one of the ways that Java supports polymorphism

Question sources

Mock exam 4 · Question 15

Question 2 | Q00983

Which of the following statement about function overloading is TRUE?

A. Overload function have same name but different parameters/type

B. Overload function is not supported

C. .
Overload function must have different return type

D. Overload function must have the same number of parameters

Hint

Compare names and parameter signatures before considering return types or access modifiers.

Answer A: Overload function have same name but different parameters/type

The test examples vary both argument counts and types while retaining test as the name.

Why each choice is right or wrong

A - Correct: Matches the definition

B - Incorrect: Overloading is explicitly supported

C - Incorrect: Return type alone is insufficient

D - Incorrect: test() and test(int,int) show that equal counts are not required

Primary note proof

Study-note section OOP12

Overloaded methods share a name but have different parameter types, counts or type order. Return type alone is insufficient in Java and C++. Overloads can also arise through inherited

methods; “same class only” is a teaching simplification. f(int), f(double) and f(int,int) are distinct, but changing a parameter name does not create an overload.

Study-note section OOP12

The compiler selects the applicable signature from the declared argument types and conversions. int arguments usually choose an exact int overload rather than a wider double overload. A return type is not part of an ordinary Java method signature.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 11 | lines 7-18

Method Overloading
• Is a process of having the same name for different methods as long as their parameter list or data types in the parameter list is different?
• The return type alone is insufficient to distinguish two versions of a method.
• Parameters are key identifiers of OM.
• Is one of the ways that Java supports polymorphism

Object Oriented Programming module | PDF page 11 | lines 24-41

```
class OverloadDemo {
void test() {
System.out.println("No
parameters"); }
void test(int a) { // Overload
test for one integer parameter.
System.out.println("a: " + a); }
void test(int a, int b) { //
Overload test for two integer
parameters.
System.out.println("a and b: " +
a + " " + b); }
double test(double a) { //
overload test for a double
parameter
System.out.println("double a: "
+ a);
return a*a; } }
```

Question sources

Computer Science Exit Exam · 2018 · Question 96

Question 3 | Q01000

Which generation of programming languages stated for high-level languages like C++, Java, Python?

A. 2nd generation

B. 3rd generation

C. 4th generation

D. None of the above

Hint

Separate language-generation classification, translation strategy and portability assumptions.

Answer B: Third generation.

OOP39 explicitly places general-purpose high-level languages such as C++, Java and Python in the third generation. This taxonomy concerns historical language classification, not whether a language is object oriented or interpreted. Thus the third-generation option applies to all the examples named in the question.

Why each choice is right or wrong

A - Incorrect: Second generation is not the category assigned to these general-purpose high-level examples in OOP39.

B - Correct: Third generation is the explicit conventional category assigned to C++, Java and Python.

C - Incorrect: Fourth generation does not match the explicit third-generation classification of the named examples.

D - Incorrect: The third-generation option is present, so none of the above cannot be correct.

Primary note proof

Study-note section OOP39

> Under the conventional introductory generation taxonomy, general-purpose high-level languages such as C++, Java, Python and Perl are third generation. The taxonomy is separate from object orientation and from compilation versus interpretation. Java source normally compiles to JVM bytecode, and a compatible JVM plus libraries supports portability across systems.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 17-29

> The Java compiler translates the program
> into an intermediate language called Java
> bytecodes.

Java bytecodes are platform-independent.
Guaranteed to be Write Once, Run
Anywhere.
There are different Java interpreters for
different platforms.
Java is: + Object Oriented + Platform
Independent
+ Simple + Secure
+ Portable +
Multithreaded + Robust

Question sources

National Exit Exam practice questions · Question 14

Computer Science Exit Exam practice collection · Question 84

Question 4 | Q01143

In which generation that classifies and categorizes high-level language like C++, JAVA, Python, Perl.

A. a | First Generation Language x

B. i? Second Generation Language x

C. |@| Third Generation Language oS

D. | Fourth Generation Language x

Hint

Separate language-generation classification, translation strategy and portability assumptions.

Answer C: Third Generation Language.

C++, Java, Python and Perl are third-generation languages in this historical classification. Compilation versus interpretation and object orientation do not alter that historical classification.

Why each choice is right or wrong

A - Incorrect: First generation does not match the third-generation classification of the named high-level languages.

B - Incorrect: Second generation does not match the conventional category OOP39 assigns to these high-level examples.

C - Correct: The named general-purpose high-level languages are explicitly classified as third generation in OOP39.

D - Incorrect: Fourth generation is not the classification given for these examples in the complete note.

Primary note proof

Study-note section OOP39

> Under the conventional introductory generation taxonomy, general-purpose high-level languages such as C++, Java, Python and Perl are third generation. The taxonomy is separate from object orientation and from compilation versus interpretation. Java source normally compiles to JVM bytecode, and a compatible JVM plus libraries supports portability across systems.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 17-29

> The Java compiler translates the program
> into an intermediate language called Java
> bytecodes.
> Java bytecodes are platform-independent.
> Guaranteed to be Write Once, Run
> Anywhere.
> There are different Java interpreters for
> different platforms.
> Java is: + Object Oriented + Platform
> Independent
> + Simple + Secure
> + Portable +
> Multithreaded + Robust

Question sources

Computer Science Exit Exam practice questions · Question 4

Question 5 | SM0073

What is the output of the following Java fragment code? int n = 6, m = 15, p = 3; n -= m; n *= p; System.out.print(n);

A. 27

B. 3

C. 24

D. -27

Hint

Respect operator grouping and operand types, then trace each operation in evaluation order.

Answer D: -27.

Start with n=6. The subtraction assignment n-=m computes 6-15, leaving n=-9. The multiplication assignment then uses that updated value: -9*3=-27. OOP23 includes this exact trace. The sign cannot become positive because only subtraction followed by multiplication by positive three occurs.

Why each choice is right or wrong

A - Incorrect: 27 discards the negative sign produced when 15 is subtracted from 6.

B - Incorrect: Three is the unchanged multiplier p, not the updated value of n that is printed.

C - Incorrect: 24 does not follow either the subtraction result -9 or its subsequent multiplication by 3.

D - Correct: The ordered updates yield n=-9 and then n=-27, which System.out.print outputs.

Primary note proof

Study-note section OOP23

> Java evaluates operands left to right while respecting precedence. With n=6,m=15,p=3, n-=m then n*=p leaves -27.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Question sources

MOE Computer Science Exit Exam · Question 21-1

National Computer Science Exit Exam · Question 77

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 76

Question 6 | SM0098

What is the output of the following Java fragment code? int[] list = new int[4]; int sum = 0; for(int i = 0; i < list.length; i++) { list[i] = i * 3; sum += list[i]; } System.out.print(sum);

A. 18

B. 6

C. 3

D. 36

Hint

Trace every element or API argument and distinguish copying values from aliasing a reference.

Answer A: 18.

The length-four array is visited at indices 0,1,2,3. Assigning i*3 stores 0,3,6,9. Starting sum at zero and adding each element gives 18, exactly the example in OOP27. The loop processes four elements, not merely the first one or an uninitialized default array.

Why each choice is right or wrong

A - Correct: All four assigned elements contribute: 0+3+6+9 equals the printed sum 18.

B - Incorrect: Six is one element's value, not the sum after all four iterations.

C - Incorrect: Three is the element assigned at index 1, rather than the full accumulator result.

D - Incorrect: 36 doubles the actual total; each of the four elements is added only once.

Primary note proof

Study-note section OOP27

> For int[] a=new int[4], setting a[i]=i*3 and accumulating yields 0+3+6+9=18.

Study-note section OOP26

> Valid indices are 0..a.length-1; an invalid access throws ArrayIndexOutOfBoundsException.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)

• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Object Oriented Programming module | PDF page 7 | lines 26-41

Iteration: The for(), while() and do...while() statements
4. Jumping Statements: to jump out of loops and to control other areas of program java uses break and continue statements.
The break statement: used to exit a loop
The Continue Statement: used to force program to go back to the top of a loop
Labels: Java does not include a goto statement.
Instead of goto, Java allows you to combine break and continue with a label.

Question sources

MOE Computer Science Exit Exam · Question 28-1

National Computer Science Exit Exam · Question 42

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 79

Question 7 | SS178

Consider the following program:

```
import java.util.Scanner;
public class Test {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.println("enter choice");
int choice = scanner.nextInt();
switch (choice) {
case 1: System.out.println("Hello");
case 2: System.out.println("you");
break;
default: System.out.println("welcome");
}
System.out.println("Goodbye");
}
}
```

After entering 1, what lines are printed following the initial “enter choice” prompt? Preserve letter case.

A. Hello
You welcome

B. Hello
You
Welcome
Goodbye

C. Hello
Goodbye

D. Hello
you
Goodbye

Hint

Trace condition tests, control transfers and printing as separate steps.

Answer D: Hello you Goodbye

Input 1 selects case 1, which prints Hello but has no break. Execution falls through to case 2 and prints you. Its break exits the switch before default, so welcome does not print. The final println outside the switch prints Goodbye. The requested output excludes the initial input prompt and preserves exact letter case.

Why each choice is right or wrong

A - Incorrect: This incorrectly includes welcome and omits the final Goodbye.

B - Incorrect: Default is skipped after case 2 breaks, so this includes an extra line.

C - Incorrect: Fall-through executes case 2, so omitting you is wrong.

D - Correct: The exact post-prompt lines are Hello, you and Goodbye.

Primary note proof

Study-note section OOP25

> break exits the loop; continue skips its remaining body and then runs a for update. Traditional colon-style switch labels fall through without break. Choice 1 printing Hello then case 2 printing you and breaking is followed by Goodbye outside the switch; default is skipped.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 51-70

> 2.3. Source file declaration
> rules
> • These rules are essential when declaring
> classes, import statements and package
> statements in a source file.
> 1. There can be only one public
> class per source file.
> 2. A source file can have multiple
> nonpublic classes.
> 3. The public class name should be
> the name of the source file as
> well which should be appended
> by .java at the end.
> For example: the class name is public class
> Employee{} then the source file should be
> as Employee.java.
> • If the class is defined inside a
> package, then the package statement
> should be the first statement in the
> source file.

Object Oriented Programming module | PDF page 7 | lines 26-41

> Iteration: The for(), while() and
> do...while() statements
> 4. Jumping Statements: to jump out
> of loops and to control other areas of
> program java uses break and
> continue statements.
> The break statement: used to exit a
> loop
> The Continue Statement: used to
> force program to go back to the top

of a loop
Labels: Java does not include a goto statement.
Instead of goto, Java allows you to combine break and continue with a label.

Question sources

Computer Science model exam collection · Question 85

Computer Science model exam collection

Question 8 | Q00035

Which one of the following is false regarding Exceptions in Java?

A. We cannot create our own exception classes by extending Exception class

B. Exception class describes errors caused by your program and external circumstances

C. Runtime Exception class describes programming errors such as bad casting and numeric errors

D. All Java exception classes inherit directly or indirectly from Throwable class

Hint

Separate source errors from runtime exceptions and determine the hierarchy and checked obligation.

Answer A: We cannot create our own exception classes by extending Exception class

The prohibition on user-defined exceptions is false: an exception class can extend Throwable or a subclass. Its examples describe input, file and resource failures and runtime programming bugs.

Why each choice is right or wrong

A - Correct: Contradicts the explicit permission to define exception classes

B - Incorrect: Consistent with user, programmer and resource failures

C - Incorrect: Consistent with runtime programming bugs, though individual bad-cast examples are not enumerated

D - Incorrect: Matches the Throwable ancestry rule

Primary note proof

Study-note section OOP29

Throwable is the root of throwable values, with Exception and Error branches.
RuntimeException and Error subclasses are unchecked; other throwable classes are checked.
IOException, FileNotFoundException and SQLException are standard checked examples.
ArithmeticException, NullPointerException, ClassCastException and array bounds exceptions are

unchecked. A missing semicolon or undeclared variable is a compile-time source error, not a checked exception.

Study-note section OOP29

A custom exception can extend Exception or RuntimeException.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 27 | lines 20-41

• An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
• An exception is an indication of a problem that occurs during a program‟s execution.
• An exception is a runtime error.
• A Java exception is an instance of a class derived from Throwable.
• The Throwable class is contained in the java.lang package, and subclasses of Throwable are contained in various packages.
• Errors related to GUI components are included in the java.awt package; numeric exceptions are included in the java.lang package because they are related to the java.lang.Number class.
• You can create your own exception classes by extending Throwable or a subclass of Throwable.

Object Oriented Programming module | PDF page 27 | lines 42-54

• Exceptions occur for various reasons.
E.g.
• A user has entered invalid data.
• A file that needs to be opened cannot be found.
• A network connection has been lost in the middle of communications or the JVM has run out of memory.
• Some of these exceptions are caused by:
• User error,
• Programmer error,
• Physical resources that have failed in some manner.

Object Oriented Programming module | PDF page 28 | lines 65-79

✓ Unchecked exceptions: Is an exception that occurs at the time of execution, these are also called as Runtime Exceptions.
✓ These include programming bugs, such as logic errors or improper use of an API.
✓ Runtime exceptions are ignored at the time of compilation.
✓ For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an ArrayIndexOutOfBoundsExceptione xception occurs.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 35

Mock exam 4 · Question 30

Question 9 | Q00067

Assuming there is a rectangle object, what getArea, getPerimeter can represent?

A. Attribute

B. Property

C. Data field

D. Behavior

Hint

Distinguish an object's stored state, operations, identity and class definition.

Answer D: Behavior

getArea and getPerimeter describe operations performed by a rectangle, so they are behavior rather than its stored state.

Why each choice is right or wrong

A - Incorrect: An attribute stores state rather than performing this operation

B - Incorrect: Property here would mean stored/descriptive state, not the two operations

C - Incorrect: A data field stores a value; these are methods

D - Correct: Methods perform operations and express characteristic behavior

Primary note proof

Study-note section OOP01

An object has identity, state and behavior. State is stored in fields; behavior is provided by methods. A Rectangle object may hold width and height and answer getArea() or getPerimeter(). Two objects can have equal fields and still be distinct objects. A class defines their common structure and behavior; an object is a particular instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 2-18

• Are the basic runtime entities in
an OO program.
• Are identifiable things or
individuals?
• Are of significance to the system
• Have states reflected in the
attributes (data) held about
the object
• Have characteristic behaviour
(i.e. functionality)
ह. Class:
• Is a blueprint for an object.
• is the definition of a set of objects
• which are all similar in terms of their
attributes
• A class consists of a class name, data
and methods.

Object Oriented Programming module | PDF page 10 | lines 62-72

Methods
• Are members of classes and do
operations.
• Allows different classes to
communicate each other.
• E.g. type name(parameter-
list)
{ // body of method
//....
return value; // if
type is not void

Question sources

MOE Computer Science Exit Exam · 2016 · Question 67

Question 10 | Q00068

Which one of the following is false about abstract classes and abstract methods?

A. We can create instances of abstract classes using the new operator

B. Implementation of abstract methods is provided by the subclasses

C. A class that contains abstract methods must be defined as abstract

D. Abstract methods are defined without implementation in the superclass

Hint

Check whether a class is concrete and whether every inherited abstract contract has an implementation.

Answer A: We can create instances of abstract classes using the new operator

The abstract-class example permits a concrete method but forbids new A(); subclasses supply the body for callme(). Thus instantiation is the false claim.

Why each choice is right or wrong

A - Correct: Direct new instantiation is forbidden

B - Incorrect: B supplies the implementation of callme in the example

C - Incorrect: An abstract method is an unimplemented requirement of an abstract class in this model

D - Incorrect: The example declares abstract void callme() without a body

Primary note proof

Study-note section OOP18

> An abstract class cannot be directly instantiated but can define fields, constructors, abstract methods and concrete methods. It need not contain an abstract method. An abstract method has no body; abstract and final conflict because an operation cannot require overriding while forbidding it.

Study-note section OOP18

> A concrete subclass must provide or inherit implementations of all required abstract methods; otherwise it must itself be abstract.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 23 | lines 57-81

> Abstract Classes
> • A classes declared with abstract
> keyword and can‟t be instantiated.
> • Used to hide meaningless methods:
> • There can be no objects of an abstract

class. .:.
• An abstract class cannot be directly instantiated with the new operator.
• You cannot declare abstract constructors, or abstract static methods
• Syntax: abstract class class_name
Example:
abstract class A {
abstract void callme();
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
}}
class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
}}

Object Oriented Programming module | PDF page 24 | lines 63-73

Summery:
Abstract:
• Classes: declared with abstract keyword and can‟t be instantiated.
• Methods: without body (no implementation)
Final:
• Class: declared as final prevents inheritance.
• Method: declared as final prevents overriding that method.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 68

Question 11 | Q00070

Which operator is used to create an array?

A. new

B. declare

C. static

D. construct

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer A: new.

OOP26 shows int[] a=new int[5] as valid array creation. Here new is the allocation operator that creates the array object. The note also allows brace initializers without spelling new in declarations, so the answer identifies the operator rather than asserting that every array initializer must explicitly contain it.

Why each choice is right or wrong

A - Correct: new is the operator used in the note's explicit array-allocation form new int[5].

B - Incorrect: declare is not the allocation operator in the Java array-creation syntax shown in OOP26.

C - Incorrect: static controls class membership; it does not create an array object by itself.

D - Incorrect: construct is not the Java array-allocation operator used in the provided creation syntax.

Primary note proof

Study-note section OOP26

> int[] a=new int[5]; and int a[]={1,2,3}; are valid. int[] a={}; creates a zero-length array; explicit new is unnecessary in a declaration initializer.

Study-note section OOP07

> static gives one field per loaded class, shared by its instances.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 8 | lines 43-55

> • a class declaration only creates a template; it does not create an actual object.
> • To actually create a Dog object, use the second statement.
> • Dog myDog; // Creates only the template
> • myDog = new Dog(); // create a Dog object called myDog
> • After this statement executes, myDog will be an instance of Dog.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 70

Computer Science Exit Exam practice questions · Question 37

Question 12 | Q00072

What does this valid Java loop print? Ignore only the trailing space.

```
for (int i = 2; i <= 11; i = i + 2) {
if (i % 4 == 0)
System.out.print(i * i + " ");
}
```

A. 16 64

B. 4163664100

C. 16644864

D. 4816644

Hint

Trace condition tests, control transfers and printing as separate steps.

Answer A: 16 64

The loop visits 2, 4, 6, 8 and 10. Only 4 and 8 have remainder zero when divided by 4, so their squares are the only values printed: 16 and 64. The string space separates the values. With valid Java capitalization, operators and parentheses, and ignoring the final space, the loop prints 16 64.

Why each choice is right or wrong

A - Correct: Only the qualifying values 4 and 8 print their squares, giving 16 64.

B - Incorrect: This includes squares from iterations where i is not divisible by four.

C - Incorrect: This adds values not printed by the two qualifying iterations.

D - Incorrect: This does not match the squares of 4 and 8.

Primary note proof

Study-note section OOP25

> while and for pre-test and may execute zero times; do-while post-tests and executes once.

Study-note section OOP23

> Java evaluates operands left to right while respecting precedence.

Study-note section OOP24

> Once a String operand is encountered, + concatenates in left-to-right groups: 1+2+"x" gives "3x", while "x"+1+2 gives "x12".

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Object Oriented Programming module | PDF page 7 | lines 26-41

> Iteration: The for(), while() and do...while() statements
> 4. Jumping Statements: to jump out of loops and to control other areas of program java uses break and continue statements.
> The break statement: used to exit a loop
> The Continue Statement: used to force program to go back to the top of a loop
> Labels: Java does not include a goto statement.

Instead of goto, Java allows you to combine break and continue with a label.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 72

Question 13 | Q00127

Which valid Java declaration reproduces a ragged table with first row 12, -9, 8; second row 7, 14; and third row -32, -1, 0? The second row has two elements, not a third element containing zero.

A. double[][] table = {12, -9, 8, 7, 14, -32, -1, 0};

B. double[][] table = { {12, -9, 8}, {7,14,0}, {-32,-1,0}};

C. double[][] table = { { 12,-9,8} {7,14} {-32,-1,0}};

D. double[][] table = { {12, -9, 8},{7, 14},{-32, -1, 0} };

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer D: Use rows {12,-9,8}, {7,14}, {-32,-1,0}.

Java represents a two-dimensional array as an array of row arrays, so the rows may have different lengths. The source table has lengths 3, 2 and 3. D supplies exactly those rows with comma-separated nested initializers. B is syntactically valid but adds an unwanted middle-row zero. A supplies scalar values where row arrays are needed, and C omits commas between its row initializers.

Why each choice is right or wrong

A - Incorrect: The declared double[][] requires row-array elements; a flat list of numeric scalars does not supply them.

B - Incorrect: Its second row contains an extra zero and therefore does not match the two-element row.

C - Incorrect: Missing commas between the row initializers make this declaration syntactically incorrect.

D - Correct: Its nested comma-separated rows match the required values and lengths 3, 2 and 3.

Primary note proof

Study-note section OOP26

int[] a=new int[5]; and int a[]={1,2,3}; are valid. int[] a={}; creates a zero-length array; explicit new is unnecessary in a declaration initializer.

Study-note section OOP26

Valid indices are 0..a.length-1; an invalid access throws ArrayIndexOutOfBoundsException. Java two-dimensional arrays are arrays of arrays and may be ragged: {{12,-9,8},{7,14},{-32,-1,0}} has row lengths 3,2,3. Java does not expose a portable physical contiguity guarantee, so a language-level answer should not rely on raw addresses.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java
(byte, short, int, long, float, double,
char & Boolean)
2. Reference types: Arrays, classes,
and interfaces, String)
• Note: Boolean type has a value true or
false (there is no conversion b/n Boolean
and other types).
• All non-primitive types are reference
types, so all class types are reference
types.
• All integers in Java are signed, Java
doesn‟t support unsigned integers.

Question sources

Addis Ababa University mock exam · Question 29

Question 14 | Q00128

Which members are good candidates for static? PI is a shared mathematical constant; the salary display reads the receiver’s salary; the hypotenuse calculation uses only side-length parameters; each Student has its own name. Select all that apply.

A. The PI field of a circle class

B. A method that displays the salary of an Employee class

C. A method that calculates the hypotenuse of a right-angle triangle class

D. A name field of a Student class

E. All except B

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer A, C: The PI field of a circle class; A method that calculates the hypotenuse of a right-angle triangle class

PI is one shared mathematical constant, so a static final field suits it. A hypotenuse utility that uses only parameters also needs no current object and can be static. Reading an employee receiver's salary requires that instance, and each student's name belongs to its own object. The aggregate option includes the instance name field, so it is not a correct complete selection.

Why each choice is right or wrong

A - Correct: PI is shared across circles and is a suitable static final constant.

B - Incorrect: The specified display reads receiver-specific salary state and should be an instance method.

C - Correct: The specified calculation uses only parameters and is suitable as a static utility.

D - Incorrect: Each student's separate name should be an instance field.

E - Incorrect: All except B still includes the per-student name field, so this set is wrong.

Primary note proof

Study-note section OOP07

> static gives one field per loaded class, shared by its instances. If count starts at zero and every constructor increments it, three creations leave count=3. Prefer ClassName.count and ClassName.utility(). A static method has no current instance, but can still use instance members through an explicit object reference.

Study-note section OOP07

> A shared mathematical PI is a good static final candidate; a student name normally belongs to an instance. A calculation using only parameters may be static.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 5 | lines 39-61

> C. Class/Static variables
> • Are declared with the static keyword in a class, but outside a method, constructor or a block.
> • There would only be one copy of each class variable per class, regardless of how many objects are created from it.
> • Are rarely used other than being declared as constants. (public static final)
> • Are stored in the static memory.
> • Static variables are created when the program starts and destroyed

when the program stops.
• Visibility is similar to instance variables. However, most static variables are declared public
• Default values are same as instance variables.
• Static variables can be accessed by calling with the class name ClassName.VariableName.

Object Oriented Programming module | PDF page 14 | lines 2-19

Static methods and Final Variables
• A static method can access only static data.
• A static method can call only other static data (Static methods & variables), but can not call a non-static data from it.
• Static method is belongs to the class and not to the object.
• A static method can be accessed directly by the class name and doesn‟t need any object.
• i.e ClassName.methodName(args)
• A static method cannot refer to "this" or "super" keywords in anyway
• But can be overloaded since they are resolved using static binding by compiler at compile time.

Question sources

Addis Ababa University mock exam · Question 30

Question 15 | Q00129

Here is the general syntax for method definition:
accessModifier returnType methodName(parameterList){
Java statements
return returnValue;
}
What is true for the accessModifier?

A. It must always be private or public.

B. It can be omitted, but if not omitted it must be private or public.

C. It can be omitted, but if not omitted there are several choices, including private and public

D. The access modifier must agree with the type of the return value.

E. All of the above

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer C: It can be omitted, but if not omitted there are several choices, including private and public

The access table includes default/no modifier, private, public and protected. Access controls visibility and has no dependency on return type.

Why each choice is right or wrong

A - Incorrect: Excludes protected and default

B - Incorrect: Allows omission but incorrectly excludes protected

C - Correct: Allows omission and the several listed access levels

D - Incorrect: Return type and visibility have different roles

E - Incorrect: A, B and D are false

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible. Top-level classes/interfaces may be public or package-private, not protected/private. Member types have additional visibility possibilities.

Study-note section OOP11

> A method has modifiers, return type, name, parameter list and body: public int getAge(){return age;}.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

Access Specifiers
Access level modifiers determine whether other classes can use a particular field or invoke a particular method.
Their are four types access levels:
1. No Modifier(Default): Visible to the package(source file), and it is called package-private
2. Private: Visible to the class only.
3. Public: Visible to the world.
4. Protected: Visible to the package and all subclasses.

Question sources

Addis Ababa University mock exam · Question 31

Question 16 | Q00130

What is the output of the following Java 17 code?

```
class FinalExam{
int num1;
public FinalExam() {
num1 = 4;
}
void display() {
System.out.println(num1) ;
}
}
class Final1 extends FinalExam{
int num=3;
public Final1() {
this(5);
num1 = num;
}
public Final1(int num) {
this.num = num * num;
}
}
public class Main{
public static void main(String[] args){
Final1 f = new Final1();
f.display();
}
}
```

A. 4

B. 25

C. 12

D. 20

Hint

Trace constructor delegation and initialization in the specified Java version.

Answer B: 25

Constructing Final1 first runs the parent constructor, setting num1 to 4. Subclass initialization sets num to 3. The delegated Final1(int) body receives 5 and sets num to 25. Control returns to the no-argument body, which assigns num1=num, replacing 4 with 25. display then prints inherited num1. Consistent subclass and constructor names make this an executable Java 17 trace.

Why each choice is right or wrong

A - Incorrect: 4 is the parent's initial num1 value, overwritten by the no-argument subclass body.

B - Correct: The delegated constructor sets num=25 and the caller copies it to num1.

C - Incorrect: 12 is not produced by squaring the delegated argument or by the assignments.

D - Incorrect: 20 is not produced by the constructor chain.

Primary note proof

Study-note section OOP10

> Under Java 17 rules, an explicit this(args) or super(args) invocation must be the first constructor statement. this invokes another constructor in the same class; super invokes the immediate superclass constructor. A constructor cannot explicitly invoke both. If there is no explicit delegation, an implicit super() is inserted; a matching accessible no-argument parent constructor must exist.

Study-note section OOP10

> Trace parent construction first, then subclass field initializers and instance blocks in order, then the relevant subclass constructor body. With consistent class and constructor names and this(5) chaining in the FinalExam example, the parent sets num1=4, the subclass initializes num=3, this(5) sets num=25, and the no-arg body sets num1=num, leaving 25.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 12 | lines 24-47

> Constructors
> • Performs automatic initialization of objects. .:.
> • Initializes an object immediately upon creation.
> • Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
> • The default constructor automatically initializes all instance variables to zero and all object references to NULL.
> • The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
> • Once you define your own constructor, the default constructor is no longer used.
> • Syntax:
> • Have the same name with their class name.
> • Have not return type, not even void.

Object Oriented Programming module | PDF page 19 | lines 43-54

When Constructors Are Called?
• In a class hierarchy, constructors are called in order of derivation, from superclass to subclass.
• Further, since super( ) must be the first statement executed in a subclass‟ constructor, this order is the same whether or not super( ) is used.
• If super( ) is not used, then the default or parameter less constructor of each superclass will be executed.

Question sources

Addis Ababa University mock exam · Question 32

Question 17 | Q00131

Which one of the following is important to implement dynamic Polymorphism?

A. Polymorphic variable

B. Method overriding

C. Inheritance

D. All of the above

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer D: All of the above

Dynamic dispatch uses an inheritance hierarchy, an overridden method and a superclass reference pointing at a subclass object. The question's polymorphic variable is that reference.

Why each choice is right or wrong

A - Incorrect: The superclass reference can hold a child object

B - Incorrect: Overriding supplies the subclass behavior

C - Incorrect: Inheritance supplies the superclass/subclass relation

D - Correct: All three participate in the cited dispatch model

Primary note proof

Study-note section OOP15

For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile. If fetch() exists only in Dog, a.fetch() fails compilation. For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Study-note section OOP14

A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype.

Study-note section OOP13

class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 22 | lines 37-53

• Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
• Dynamic method dispatch is important because this is how Java implements run-time polymorphism
• When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs.
• Thus, this determination is made at run time.
• Method Overriding is an example of runtime polymorphism.

Object Oriented Programming module | PDF page 25 | lines 17-37

• Polymorphism is the ability of an object to take on many forms.
• The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
• Any Java object that can pass more than one IS-A test is considered to be polymorphic.
• In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

• Polymorphism is derived from 2 Greek words: poly and morphs.
• The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
• polymorphism is the capability of an action or method to do different things based on the object that it is acting upon.

Object Oriented Programming module | PDF page 15 | lines 23-53

What is Inheritance?
• Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Question sources

Addis Ababa University mock exam · Question 33

Question 18 | Q00132

An accessible parent declares public abstract int value(); and its direct child is non-abstract with no inherited implementation of that method. What must the child do?

A. A child must override an abstract method inherited from its parent by defining a method with the same signature and same return type

B. A child must define an additional method similar to the one inherited from its parent by defining a method with the same signature and different return type.

C. Achild must not define any method with the same signature as the parent's abstract method.

D. A non-abstract child must define an abstract method with the same signature and same return type as the parent's abstract method.

E. All of the above

Hint

Check whether a class is concrete and whether every inherited abstract contract has an implementation.

Answer A: A child must override an abstract method inherited from its parent by defining a method with the same signature and same return type

A concrete child must implement every outstanding abstract method. Here the inherited contract has no parameters and a primitive int return, so the child must supply a public concrete implementation of that same signature and return type. The covariant reference-return exception does not apply to primitive int. A different return type, refusing to implement the method, or declaring it abstract leaves the concrete-class requirement unsatisfied.

Why each choice is right or wrong

A - Correct: A concrete matching no-argument int implementation, with the required public access, satisfies the inherited contract.

B - Incorrect: A different return type cannot implement this primitive int-returning contract.

C - Incorrect: Omitting a matching implementation leaves an abstract obligation in the concrete child.

D - Incorrect: An abstract declaration supplies no implementation and cannot satisfy the concrete child’s obligation.

E - Incorrect: B, C and D fail the stated implementation requirement, so the aggregate cannot be correct.

Primary note proof

Study-note section OOP18

> A concrete subclass must provide or inherit implementations of all required abstract methods; otherwise it must itself be abstract. If parent declares abstract String welcome(), a String welcome() implementation is required; int welcome() is incompatible, and welcome(int) merely

adds an overload. A valid class can contain both the required implementation and an unrelated overload.

Study-note section OOP14

A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype. Visibility cannot become more restrictive, and checked exceptions cannot be broadened incompatibly.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 23 | lines 57-81

Abstract Classes
• A classes declared with abstract
keyword and can‟t be instantiated.
• Used to hide meaningless methods:
• There can be no objects of an abstract
class. .:.
• An abstract class cannot be directly
instantiated with the new operator.
• You cannot declare abstract
constructors, or abstract static methods
• Syntax: abstract class class_name
Example:
abstract class A {
abstract void callme();
// concrete methods are still
allowed in abstract classes
void callmetoo() {
System.out.println("This is a
concrete method.");
}}
class B extends A {
void callme() {
System.out.println("B's
implementation of callme.");
}}

Object Oriented Programming module | PDF page 11 | lines 55-67

Method Overriding
In a class hierarchy, when a method in a
subclass has the same name and type
signature as a method in its superclass, this
is called method Overriding.
• the method in the subclass is said to
override the method in the
superclass.

• The Method in the super class is known as an overridden method which is called within a subclass.
• The version of the method defined by the superclass will be hidden.

Object Oriented Programming module | PDF page 31 | lines 9-30

• It provides total abstraction;
• Means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default.
• A class that implement interface must implement all the methods declared in the interface.
• If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
Rules of Interface
• Variables declared in an interface are public, static & final by default.
• Java does not allow you to extend more than one class, However you can implement more than one interfaces in your class.
• class implements interface but an interface extends another interfaces.

Question sources

Addis Ababa University mock exam · Question 34

Mock exam 3 · Question 18

Question 19 | Q00133

Which of the following is true?

A. A child class can extend a parent or implement an interface, but not do both.

B. A child class can extend just one parent and can implement just one interface.

C. A child class can extend just one parent and can implement zero or more interfaces.

D. A child class can extend zero or more parents, and can implement zero or more interfaces.

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer C: A child class can extend just one parent and can implement zero or more interfaces.

A Java class can extend one class and implement multiple interfaces.

Why each choice is right or wrong

A - Incorrect: No exclusive choice between extending and implementing is stated

B - Incorrect: Artificially limits interfaces to one

C - Correct: One direct superclass and zero or more interfaces matches the rule

D - Incorrect: Multiple direct parent classes are forbidden

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent. A class extends one direct class and can implement multiple interfaces. Single, multilevel and hierarchical inheritance mean one parent, a chain, and several children of one parent. Java does not permit class Dog extends Animal,Mammal.

Study-note section OOP19

> A class can extend a superclass and implement zero or more interfaces.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 31 | lines 9-30

> • It provides total abstraction;
> • Means all the methods in interface
> are declared with empty body and
> are public and all fields are public,
> static and final by default.
> • A class that implement interface
> must implement all the methods
> declared in the interface.
> • If a class implements an interface
> and does not provide method bodies
> for all functions specified in the
> interface, then class must be declared
> abstract.
> Rules of Interface
> • Variables declared in an interface are
> public, static & final by default.

• Java does not allow you to extend more than one class, However you can implement more than one interfaces in your class.
• class implements interface but an interface extends another interfaces.

Question sources

Addis Ababa University mock exam · Question 35

Mock exam 4 · Question 27

Question 20 | Q00224

What will be the output of the program?

```
try {
int x = 0;
int y = 5 / x;
} catch (Exception e) {
System.out.println("Exception");
} catch (ArithmeticException ae) {
System.out.println("Arithmetic Exception");
}
System.out.println("finished");
```

A. Arithmetic Exception

B. finished

C. Exception

D. Compilation fails.

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer D: Compilation fails because the subtype catch is unreachable.

Exception is caught before its subclass ArithmeticException. OOP30 says that a general catch before a subtype catch makes the latter unreachable and prevents compilation. The program therefore never executes the division or either print statement. Reasoning about which handler would run cannot bypass this compile-time restriction.

Why each choice is right or wrong

A - Incorrect: The ArithmeticException handler is unreachable at compile time, so it cannot print its message.

B - Incorrect: Compilation fails before execution; the statement following the catches never prints finished.

C - Incorrect: Although Exception could catch this runtime type, the illegal later catch prevents any execution.

D - Correct: The earlier Exception catch subsumes ArithmeticException, making the later handler unreachable and the program uncompilable.

Primary note proof

Study-note section OOP30

> Catch blocks are considered in source order. Place FileNotFoundException before IOException and IOException before Exception; a general catch before a subtype catch makes that later handler unreachable and prevents compilation.

Study-note section OOP29

> Throwable is the root of throwable values, with Exception and Error branches. RuntimeException and Error subclasses are unchecked; other throwable classes are checked. IOException, FileNotFoundException and SQLException are standard checked examples. ArithmeticException, NullPointerException, ClassCastException and array bounds exceptions are unchecked.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 29 | lines 11-22

> Using try and catch...
> General form of try/catch:
> Try{
> //block of code to monitor for
> errors
> }
> Catch(ExceptionType1 exOb) {
> //handler for ExceptionType1
> }
> Catch(ExceptionType2 exOb) {
> //handler for ExceptionType2
> }

Object Oriented Programming module | PDF page 27 | lines 20-41

> • An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
> • An exception is an indication of a problem that occurs during a program‟s execution.
> • An exception is a runtime error.
> • A Java exception is an instance of a class derived from Throwable.

• The Throwable class is contained in the java.lang package, and subclasses of Throwable are contained in various packages.
• Errors related to GUI components are included in the java.awt package; numeric exceptions are included in the java.lang package because they are related to the java.lang.Number class.
• You can create your own exception classes by extending Throwable or a subclass of Throwable.

Question sources

Mock exam 4 · Question 26

Question 21 | Q00226

Which one of the following constant declaration statements is not possible in the Interface body?

A. static double PI = 3.14;

B. static final double PI = 3.14;

C. double PI = 3.14;

D. final double PI = 3.14;

E. None of the choices is correct

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer E: None of the choices is correct

All interface fields are implicitly public, static and final; writing either, both or neither of static and final does not remove the implicit modifiers. Each PI declaration also has an initializer.

Why each choice is right or wrong

A - Incorrect: static is explicit and final implicit

B - Incorrect: Both static and final are explicit

C - Incorrect: Both modifiers are implicit

D - Incorrect: final is explicit and static implicit

E - Correct: All four declarations satisfy the rule, so no listed declaration is impossible

Primary note proof

Study-note section OOP19

No interface instance fields or constructors exist; fields are implicitly public static final.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 31 | lines 9-30

• It provides total abstraction;
• Means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default.
• A class that implement interface must implement all the methods declared in the interface.
• If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
Rules of Interface
• Variables declared in an interface are public, static & final by default.
• Java does not allow you to extend more than one class, However you can implement more than one interfaces in your class.
• class implements interface but an interface extends another interfaces.

Question sources

Mock exam 4 · Question 28

Question 22 | Q00227

Which one of the following is wrong about Method Overriding?

A. Method overriding is one way of implementing polymorphism at run time.

B. Method overriding deals with methods with similar signatures in the base and derived classes.

C. Method overriding deals with implementing similar operations for different objects.

D. Method overriding is possible between any two classes.

Hint

Check inheritance, signature compatibility, visibility and eligibility for an override.

Answer D: Method overriding is possible between any two classes.

Overriding is a superclass/subclass relationship with matching method signatures and supports runtime dispatch, not a relationship between arbitrary unrelated classes.

Why each choice is right or wrong

A - Incorrect: Runtime polymorphism is an explicit purpose

B - Incorrect: Matches the hierarchy and signature rule

C - Incorrect: Different subclass implementations can specialize the inherited operation

D - Correct: Unrelated arbitrary classes do not establish overriding

Primary note proof

Study-note section OOP14

> A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype.

Study-note section OOP14

> Merely reusing a name in two unrelated classes does not establish overriding.

Study-note section OOP15

> For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 11 | lines 55-67

> Method Overriding
> In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, this is called method Overriding.
> • the method in the subclass is said to override the method in the superclass.
> • The Method in the super class is known as an overridden method which is called within a subclass.
> • The version of the method defined by the superclass will be hidden.

Object Oriented Programming module | PDF page 22 | lines 37-53

> • Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

• Dynamic method dispatch is important because this is how Java implements run-time polymorphism
• When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs.
• Thus, this determination is made at run time.
• Method Overriding is an example of runtime polymorphism.

Question sources

Mock exam 4 · Question 29

Question 23 | Q00277

Public classes A, B and C are in three different packages, with C extending B and B extending A. From code declared inside C, accessing fields through this, which is the complete access set? Assume ordinary instance fields with distinct names, accessible public declaring types, and no hiding.

A. All fields declared in C, and only accessible inherited protected and public fields from A and B

B. Protected, default and public data only in C and B.

C. Any data in C and only public data in A and B.

D. Protected, default and public data only in C.

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer A: All fields declared in C, and only accessible inherited protected and public fields from A and B

Code inside C can access C’s own fields, including private ones. Across package boundaries it can also access inherited public and protected fields through its own inherited context, here represented by this. It cannot access package-private or private ancestor fields directly. Consequently A gives the complete set; the public-only choice is incomplete because qualifying protected inherited access is also permitted.

Why each choice is right or wrong

A - Correct: This includes C’s own fields and the inherited public/protected access permitted through this.

B - Incorrect: Package-private ancestor fields in different packages are not accessible, and A's inherited access is omitted.

C - Incorrect: This omits accessible inherited protected fields and therefore is not the complete set.

D - Incorrect: This omits C's own private access and permitted inherited ancestor access.

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible. Top-level classes/interfaces may be public or package-private, not protected/private. Member types have additional visibility possibilities.

Study-note section OOP08

> Across packages, a subclass accessing a protected instance member must do so through its own inherited context or an appropriate subclass-typed receiver, not any arbitrary superclass object.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

> Access Specifiers
> Access level modifiers determine whether
> other classes can use a particular field or
> invoke a particular method.
> Their are four types access levels:
> 1. No Modifier(Default): Visible to
> the package(source file), and it is
> called package-private
> 2. Private: Visible to the class only.
> 3. Public: Visible to the world.
> 4. Protected: Visible to the package
> and all subclasses.

Object Oriented Programming module | PDF page 17 | lines 4-9

> Member Access and Inheritance
> • Although a subclass includes all of
> the members of its superclass, it
> cannot access those members of the
> superclass that have been declared as
> private.

Question sources

Computer Science model exam · Question 8

University of Gondar model exam · Question 8

Question 24 | Q00278

Inside an instance method, which shown forms invoke an accessible no-argument non-static method? Methodname stands for a valid method name, Objectname for a compatible object reference, and Classname for a type name. Select all that apply.

A. Methodname()

B. Classname.methodname()

C. Objectname.methodname()

D. Variablename.methodname

Hint

Separate parameters from arguments and operations that read state from operations that change it.

Answer A, C: Methodname(); Objectname.methodname()

An unqualified call inside a suitable instance context uses implicit this, so Methodname() is valid. An object-qualified call explicitly supplies the receiver and is also valid. A type name alone cannot invoke a non-static method without an object. The final form lacks invocation parentheses and therefore does not invoke a method. These are deliberate syntax alternatives under the stated placeholder meanings.

Why each choice is right or wrong

A - Correct: The unqualified call uses implicit this in the stated instance context.

B - Incorrect: A class name alone does not provide the required instance receiver.

C - Correct: The object-qualified call supplies the instance receiver.

D - Incorrect: Without parentheses this form does not perform a method invocation.

Primary note proof

Study-note section OOP11

> A method has modifiers, return type, name, parameter list and body: public int getAge(){return age;}. A void method returns no value. Parameters name the inputs in the declaration; arguments supply values at a call. An instance method can be invoked as object.method(), or with an implicit this inside suitable instance context.

Study-note section OOP16

> Java this and super are unavailable as current-instance references in static context.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 14 | lines 2-19

Static methods and Final Variables
• A static method can access only static data.
• A static method can call only other static data (Static methods & variables), but can not call a non-static data from it.
• Static method is belongs to the class and not to the object.
• A static method can be accessed directly by the class name and doesn‟t need any object.
• i.e ClassName.methodName(args)
• A static method cannot refer to "this" or "super" keywords in anyway
• But can be overloaded since they are resolved using static binding by compiler at compile time.

Object Oriented Programming module | PDF page 11 | lines 24-41

```
class OverloadDemo {
void test() {
System.out.println("No
parameters"); }
void test(int a) { // Overload
test for one integer parameter.
System.out.println("a: " + a); }
void test(int a, int b) { //
Overload test for two integer
parameters.
System.out.println("a and b: " +
a + " " + b); }
double test(double a) { //
overload test for a double
parameter
System.out.println("double a: "
+ a);
return a*a; } }
```

Question sources

Computer Science model exam · Question 9

University of Gondar model exam · Question 9

Question 25 | Q00279

In Java, Base declares an eligible instance method m(), Sub overrides it, and an object's exact runtime class is Sub. Inside an instance method declared in Sub, what happens on an unqualified call m()? Assume no further subclass or explicit super call.

A. The Compiler will choose randomly the method to be executed

B. It will always refer to the version of that method defined by the subclass

C. Compile time error will occur

D. It will always refer to the version of that method defined by the superclass

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer B: It will always refer to the version of that method defined by the subclass

The unqualified invocation uses the current Sub object as receiver. Since m is an eligible overridden instance method and the exact runtime class is Sub, dynamic dispatch selects Sub's implementation. The compiler does not choose randomly, and the valid declaration produces no compilation error. An explicit super.m() would request the parent version, but that is excluded by the specified call expression.

Why each choice is right or wrong

A - Incorrect: Dispatch follows the language's receiver and overriding rules, not random choice.

B - Correct: The exact Sub receiver dispatches the ordinary call to Sub.m().

C - Incorrect: The stated accessible, eligible declaration and call compile.

D - Incorrect: The superclass version would require a different dispatch situation such as explicit super.m().

Primary note proof

Study-note section OOP15

> For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile. If fetch() exists only in Dog, a.fetch() fails compilation. For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Study-note section OOP16

> An unqualified instance call may dispatch dynamically; an explicit super call is different.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 11 | lines 55-67

> Method Overriding
> In a class hierarchy, when a method in a
> subclass has the same name and type

signature as a method in its superclass, this is called method Overriding.
• the method in the subclass is said to override the method in the superclass.
• The Method in the super class is known as an overridden method which is called within a subclass.
• The version of the method defined by the superclass will be hidden.

Object Oriented Programming module | PDF page 17 | lines 51-74

The super keyword
• Since encapsulation is a primary attribute of OOP, Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.
• Super has two general forms.
1. Used to access parent class instance members.
2. Used to access parent class constructor.
1. Used to access parent class instance members.
• super acts somewhat like this, except that it always refers to the superclass of the subclass in which it is used.
• Super has the form: super.member Here, member can be either a method or an instance variable.
• Super is most applicable to resolve member hiding.
• super( ) must always be the first statement executed inside a subclass‟ constructor.

Question sources

Computer Science model exam · Question 10

University of Gondar model exam · Question 10

Question 26 | Q00280

Which of the following statements not true?

A. A private member of a class cannot be accessed by the methods of the same class

B. A public member of a class can be accessed in all the packages

C. A private member of a class cannot be accessed from its derived class

D. A protected member of a class can be accessed from its derived class.

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer A: A private member of a class cannot be accessed by the methods of the same class

Private visibility is class-only, so methods of the declaring class can access that member. Public is world-visible; protected includes subclasses.

Why each choice is right or wrong

A - Correct: False: same-class methods have access

B - Incorrect: Public visibility includes other packages

C - Incorrect: A child cannot directly access a parent's private member

D - Incorrect: Protected includes subclass access

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible. Top-level classes/interfaces may be public or package-private, not protected/private. Member types have additional visibility possibilities.

Study-note section OOP08

> Across packages, a subclass accessing a protected instance member must do so through its own inherited context or an appropriate subclass-typed receiver, not any arbitrary superclass object. protected therefore does not mean “subclasses only”: unrelated classes in the same package also have access. A class’s own main method can access private fields of an instance of that class.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

> Access Specifiers
> Access level modifiers determine whether
> other classes can use a particular field or
> invoke a particular method.
> Their are four types access levels:

1. No Modifier(Default): Visible to the package(source file), and it is called package-private
2. Private: Visible to the class only.
3. Public: Visible to the world.
4. Protected: Visible to the package and all subclasses.

Object Oriented Programming module | PDF page 17 | lines 4-9

Member Access and Inheritance
• Although a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared as private.

Question sources

Computer Science model exam · Question 11

University of Gondar model exam · Question 11

Question 27 | Q00281

_____is one of the basic principles of object oriented programming that use one object in multiple forms.

A. Inheritance

B. Interface

C. Encapsulation

D. None of the above

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer D: None of the above

Taking many forms is polymorphism. It is not explicitly listed, so the none option is the correct selection.

Why each choice is right or wrong

A - Incorrect: Inheritance acquires parent properties

B - Incorrect: Interface defines a contract rather than naming the many-forms principle

C - Incorrect: Encapsulation groups and hides data

D - Correct: The principle is polymorphism

Primary note proof

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Study-note section OOP02

> Encapsulation groups related state and operations and controls access.

Study-note section OOP19

> interface Payable{double getPaymentAmount();} declares an abstract public operation. class Invoice implements Payable must provide public double getPaymentAmount(), perhaps returning getQuantity()*getPricePerItem().

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 58-64

> 2. Polymorphism:
> • The word polymorphism means the ability to take more than one form.
> • In terms of the OOP, this means that a particular operation may behave differently for different sub-classes of the same class.

Question sources

Computer Science model exam · Question 12

University of Gondar model exam · Question 12

Question 28 | Q00282

Consider the program:

public class Student {

int id;

int section;

void display() {

System.out.println("ID : " + id);
System.out.println("Section : " + section); }

public static void main(String[] args) {

/*Create an object*/ }}

Which of the following is a correct way to create an object in the main method of the above program?

A. Student stud = Student (3983,20);

B. Student stud;

C. Student stud=new Student ();

D. Student stud =new Student (3983,2);

Hint

Separate a reference declaration, object allocation, reference copying and changes to the shared object.

Answer C: Student stud=new Student ();

No constructor is declared in Student, so the default no-argument construction pattern applies. A reference declaration alone does not create an object.

Why each choice is right or wrong

A - Incorrect: Missing new and invokes an undeclared parameterized constructor

B - Incorrect: Declares only a reference

C - Correct: Uses new and the implicit no-argument constructor

D - Incorrect: No matching two-argument constructor is declared

Primary note proof

Study-note section OOP03

> Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance.

Study-note section OOP09

If a class declares no constructor, Java supplies a default constructor. If Circle(double r) is declared, new Circle() fails unless a no-argument constructor is also declared.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 8 | lines 43-55

• a class declaration only creates a template; it does not create an actual object.
• To actually create a Dog object, use the second statement.
• Dog myDog; // Creates only the template
• myDog = new Dog(); // create a Dog object called myDog
• After this statement executes, myDog will be an instance of Dog.

Object Oriented Programming module | PDF page 12 | lines 24-47

Constructors
• Performs automatic initialization of objects. .:.
• Initializes an object immediately upon creation.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
• The default constructor automatically initializes all instance variables to zero and all object references to NULL.
• The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
• Once you define your own constructor, the default constructor is no longer used.
• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Question sources

Computer Science model exam · Question 13

University of Gondar model exam · Question 13

Question 29 | Q00431

Default, public , protected, private are?

A. Access Modifier

B. Non-access Modifiers

C. Both A and B

D. It's variable

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer A: Access Modifier

The access table names no modifier/default, public, protected and private as visibility levels.

Why each choice is right or wrong

A - Correct: They control accessibility

B - Incorrect: They are visibility modifiers rather than non-access properties such as static

C - Incorrect: No basis for classifying the same list as both

D - Incorrect: They are not a variable

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible. Top-level classes/interfaces may be public or package-private, not protected/private. Member types have additional visibility possibilities.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

> Access Specifiers
> Access level modifiers determine whether
> other classes can use a particular field or
> invoke a particular method.
> Their are four types access levels:
> 1. No Modifier(Default): Visible to
> the package(source file), and it is

called package-private
2. Private: Visible to the class only.
3. Public: Visible to the world.
4. Protected: Visible to the package
and all subclasses.

Question sources

Object-Oriented Programming practice questions · Question 1

Question 30 | Q00432

Java Source Code is compiled into _____.

A. .Obj

B. Source Code

C. Bytecode

D. .Exe

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer C: Bytecode

The compiler produces intermediate Java bytecodes, explicitly described as platform-independent.

Why each choice is right or wrong

A - Incorrect: Not the compiler’s intermediate-language output

B - Incorrect: Input source is translated rather than simply reproduced

C - Correct: Matches the stated intermediate language

D - Incorrect: Not the compiler’s intermediate-language output

Primary note proof

Study-note section OOP28

Ordinary compilation uses javac Test.java, producing bytecode, then java Test launches the class.

Study-note section OOP39

Java source normally compiles to JVM bytecode, and a compatible JVM plus libraries supports portability across systems.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 17-29

The Java compiler translates the program into an intermediate language called Java bytecodes.
Java bytecodes are platform-independent.
Guaranteed to be Write Once, Run Anywhere.
There are different Java interpreters for different platforms.
Java is: + Object Oriented + Platform Independent
+ Simple + Secure
+ Portable +
Multithreaded + Robust

Question sources

Object-Oriented Programming practice questions · Question 2

Question 31 | Q00433

Single line comment starts with _____ in Java.

A. /**

B. //

C. /*

D. None of these

Hint

Check the lexical category and whether the notation names a value, a declaration or non-executed source text.

Answer B: //

// begins a single-line comment; /* and /** begin block comments.

Why each choice is right or wrong

A - Incorrect: Starts documentation-style block comments

B - Correct: The single-line delimiter

C - Incorrect: Starts a block comment

D - Incorrect: The required delimiter is listed as B

Primary note proof

Study-note section OOP-S01

> A single-line Java comment begins with // and continues to the end of that line. A block comment begins with /* and ends with */. A documentation comment begins with /** and ends with */ and can be processed by Javadoc. Comment text does not execute as program statements.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 2-4

> single line comment //, 2 multiline
> comments /* .... */ and /** ....*/.
> The las one is used by javaDoc

Question sources

Object-Oriented Programming practice questions · Question 3

Question 32 | Q00434

How to compile java code in command prompt?

A. javac filename.java

B. java filename.java

C. javac filename

D. java filename

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer A: javac filename.java

The FileNotFound example shows C:\>javac FilenotFound_Demo.java, establishing the compiler command with a .java source filename.

Why each choice is right or wrong

A - Correct: Matches the javac compilation command

B - Incorrect: Uses java rather than the demonstrated compiler

C - Incorrect: Omits the .java source suffix

D - Incorrect: Uses the runtime command name rather than javac

Primary note proof

Study-note section OOP28

> Ordinary compilation uses javac Test.java, producing bytecode, then java Test launches the class. Modern Java can also launch a source file using java Test.java; distinguish source-file launch from running a compiled class.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 28 | lines 54-64

> • If you try to compile the
> above program you will get
> exceptions like this.
> C:\>javac FilenotFound_Demo.java
> FilenotFound_Demo.java:8: error:
> unreported exception
> FileNotFoundException; must be
> caught or declared to be thrown
> FileReader fr = new
> FileReader(file);
> 1 error

Question sources

Object-Oriented Programming practice questions · Question 4

Question 33 | Q00435

An ordinary class named filename has already been compiled to filename.class, is in the current directory with no package, and has a valid main method. Which command launches that compiled class? Use class launch, not source-file mode.

A. javac filename.java

B. java filename.java

C. javac filename

D. java filename

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer D: java filename

The Java launcher takes the compiled class's name for ordinary class launch, so java filename runs the available main method. javac filename.java compiles a source file and is a separate step. Modern Java also supports source-file launch with a .java argument, but the revised question explicitly asks for the already compiled class. The compiler invocation without the source extension is not the requested launch either.

Why each choice is right or wrong

A - Incorrect: This compiles source rather than launching the existing compiled class.

B - Incorrect: This is source-file launch, excluded by the requested class-launch mode.

C - Incorrect: This is a compiler invocation rather than the requested class launch.

D - Correct: java filename launches the named compiled class.

Primary note proof

Study-note section OOP28

> Ordinary compilation uses javac Test.java, producing bytecode, then java Test launches the class. Modern Java can also launch a source file using java Test.java; distinguish source-file launch from running a compiled class.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 17-29

> The Java compiler translates the program
> into an intermediate language called Java
> bytecodes.

Java bytecodes are platform-independent.
Guaranteed to be Write Once, Run
Anywhere.
There are different Java interpreters for
different platforms.
Java is: + Object Oriented + Platform
Independent
+ Simple + Secure
+ Portable +
Multithreaded + Robust

Object Oriented Programming module | PDF page 28 | lines 54-64

• If you try to compile the
above program you will get
exceptions like this.
C:\>javac FilenotFound_Demo.java
FilenotFound_Demo.java:8: error:
unreported exception
FileNotFoundException; must be
caught or declared to be thrown
FileReader fr = new
FileReader(file);
1 error

Question sources

Object-Oriented Programming practice questions · Question 5

Question 34 | Q00436

Java is case sensitive?

A. True

B. False

C. Depends On Complier

D. May be true or false

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer A: True: Java is case sensitive.

OOP28 explicitly states that Java identifiers are case sensitive. Changing the capitalization of a name can therefore change what it denotes or make a reference unresolved. This is a language rule rather than a compiler preference, so the conditional or uncertain answers do not describe standard Java behavior.

Why each choice is right or wrong

A - Correct: True directly matches the note's explicit statement that Java identifiers are case sensitive.

B - Incorrect: False contradicts the explicit identifier case-sensitivity rule stated in OOP28.

C - Incorrect: Case sensitivity is a Java language rule, not an option depending on the compiler implementation.

D - Incorrect: The note states a definite language property rather than a true-or-false implementation-dependent possibility.

Primary note proof

Study-note section OOP28

> Java identifiers are case sensitive; $ is allowed but @ is not an identifier character.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 11-20

> ✓ Reserved keywords: class, extends, implements, for...
> ✓ Identifiers: programmers given name; grade, gRade...
> ✓ Literals (Constants values): 55, 60.78, "Hi", „M"... Java language specifies five major types of literals => integer, floating, character, string & Boolean literals

Question sources

Object-Oriented Programming practice questions · Question 6

Question 35 | Q00437

Which is the most complete description of the standard introductory naming conventions and ordinary public-top-level-class file rule in Java? Treat "should" in A and B as style advice, not a syntax requirement, and "class" in C as the public top-level class in a .java source file.

A. For all class names the first letter should be in Upper Case.

B. All method names should start with a Lower Case letter.

C. Name of the program file should exactly match the class name.

D. All of the above

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer D: All of the above

Capitalized class initials and lowercase method initials are conventional Java style, not universal syntax requirements. An ordinary public top-level class must match its source filename's base name. The scope makes A and B recommendations and C the specific file rule. Each is then true, but D is the most complete selection because it includes all three rather than one isolated part.

Why each choice is right or wrong

A - Incorrect: This is valid style advice but only one part of the complete set.

B - Incorrect: This is valid style advice but omits the other convention and file rule.

C - Incorrect: This is the file rule for the specified public top-level class but is incomplete alone.

D - Correct: All three are valid under the stated convention and file-rule scope.

Primary note proof

Study-note section OOP28

> For ordinary public top-level types, filename and type name match. Capitalized class names and lowercase method names are conventions, not universal syntax requirements.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 51-70

> 2.3. Source file declaration
> rules
> • These rules are essential when declaring
> classes, import statements and package
> statements in a source file.
> 1. There can be only one public
> class per source file.
> 2. A source file can have multiple
> nonpublic classes.
> 3. The public class name should be
> the name of the source file as
> well which should be appended
> by .java at the end.
> For example: the class name is public class
> Employee{} then the source file should be
> as Employee.java.
> • If the class is defined inside a
> package, then the package statement
> should be the first statement in the
> source file.

Object Oriented Programming module | PDF page 4 | lines 11-20

✓ Reserved keywords: class, extends, implements, for...
✓ Identifiers: programmers given name; grade, gRade...
✓ Literals (Constants values): 55, 60.78, “Hi”, „M“... Java language specifies five major types of literals => integer, floating, character, string & Boolean literals

Question sources

Object-Oriented Programming practice questions · Question 7

Question 36 | Q00438

All Java components require names. Names used for classes, variables, and methods are called?

A. Variables

B. identifiers

C. Access Modifiers

D. Java Modifiers

Hint

Check the lexical category and whether the notation names a value, a declaration or non-executed source text.

Answer B: identifiers

Identifiers are programmer-given names. Names attached to classes, variables and methods serve that role.

Why each choice is right or wrong

A - Incorrect: A variable is one named entity, not the name category

B - Correct: Matches programmer-given names

C - Incorrect: Controls visibility rather than naming

D - Incorrect: Modifiers change declarations rather than supply their names

Primary note proof

Study-note section OOP-S01

An identifier is a programmer-chosen name for an entity such as a class, method or variable. A reserved keyword has a language-defined role and cannot be used as an ordinary identifier. A

literal writes a value directly, such as an integer, floating-point number, quoted character, quoted string or Boolean value.

Study-note section OOP08

For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 11-20

✓ Reserved keywords: class, extends, implements, for...
✓ Identifiers: programmers given name; grade, gRade...
✓ Literals (Constants values): 55, 60.78, “Hi”, „M“... Java language specifies five major types of literals => integer, floating, character, string & Boolean literals

Question sources

Object-Oriented Programming practice questions · Question 8

Question 37 | Q00439

Which of the following is smallest integer data type ?

A. int

B. byte

C. short

D. long

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer B: byte

The actual PDF page-six widening diagram orders byte -> short -> int -> long -> float (and onward to double); the text says widening requires the destination to be wider than the source. Among byte, short, int and long, byte is therefore the smallest in this module diagram. The diagram is a raster/embedded visual omitted from extracted text, so the exact textual citation supplies the associated width condition rather than fabricating a diagram quotation.

Why each choice is right or wrong

A - Incorrect: int appears after byte and short in the widening diagram

B - Correct: byte is the earliest/narrowest listed integer type in the diagram

C - Incorrect: short is wider than byte in the diagram

D - Incorrect: long appears after int in the diagram

Primary note proof

Study-note section OOP06

> Java has byte, short, int, long, float, double, char and boolean. Signed integer widths are 8,16,32,64 bits respectively. byte ranges from -128 to 127; short from -32768 to 32767.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean
> and other types).
> • All non-primitive types are reference
> types, so all class types are reference
> types.
> • All integers in Java are signed, Java
> doesn‟t support unsigned integers.

Object Oriented Programming module | PDF page 6 | lines 30-36

> Two conditions for automatic
> casting
> ✓ Types are compatible
> ✓ Destination is wider
> than the source
> Widening conversion usually
> occur automatically.

Question sources

Object-Oriented Programming practice questions · Question 9

Question 38 | Q00440

Which of the following is not a primitive data type ?

A. byte

B. enum

C. short

D. int

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer B: enum

The exhaustive eight-primitive list includes byte, short and int but not enum.

Why each choice is right or wrong

A - Incorrect: Explicit primitive

B - Correct: Not among the eight primitives

C - Incorrect: Explicit primitive

D - Incorrect: Explicit primitive

Primary note proof

Study-note section OOP06

> Java has byte, short, int, long, float, double, char and boolean.

Study-note section OOP06

> enum and String are reference types.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean
> and other types).
> • All non-primitive types are reference
> types, so all class types are reference

types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 10

Question 39 | Q00441

Integer Data type does not include following primitive data type _____.

A. long

B. byte

C. short

D. double

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer D: double is not an integer primitive type.

OOP06 lists byte, short, int and long as the signed integer types with widths 8,16,32,64 bits. Of the supplied options, long, byte and short belong to that group. double represents floating-point values rather than an integer primitive, so it is the exception requested by the question.

Why each choice is right or wrong

A - Incorrect: long is the 64-bit signed integer primitive in the note's integer-width sequence.

B - Incorrect: byte is the 8-bit signed integer primitive with the stated range -128 to 127.

C - Incorrect: short is the 16-bit signed integer primitive with the stated range -32768 to 32767.

D - Correct: double belongs to the floating numeric types rather than the signed integer primitive group.

Primary note proof

Study-note section OOP06

Java has byte, short, int, long, float, double, char and boolean. Signed integer widths are 8,16,32,64 bits respectively.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 11

Question 40 | Q00442

Character data type cannot store following value.

A. Digit

B. Letter

C. Special Character

D. String

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer D: String

Character literals such as 'M' represent individual characters, while strings such as "Hi" are sequences of characters. A whole String is not one char value.

Why each choice is right or wrong

A - Incorrect: A digit can be a single character, not a multi-character string

B - Incorrect: A letter is a single character, illustrated by M

C - Incorrect: A special symbol can be one character

D - Correct: A string is a sequence rather than one character

Primary note proof

Study-note section OOP06

char is an unsigned 16-bit UTF-16 code unit and can hold a digit, letter or symbol, not a whole String.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 11-20

✓ Reserved keywords: class, extends, implements, for...
✓ Identifiers: programmers given name; grade, gRade...
✓ Literals (Constants values): 55, 60.78, “Hi”, „M“... Java language specifies five major types of literals => integer, floating, character, string & Boolean literals

Object Oriented Programming module | PDF page 9 | lines 14-16

5. String Class:- Immutable Class created to manipulate strings(Sequence of chars).

Question sources

Object-Oriented Programming practice questions · Question 12

Question 41 | Q00443

Range of Byte Data Type is _____.

A. -128 to 128

B. -127 to 127

C. -127 to 128

D. -128 to 127

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer D: -128 to 127, inclusive.

OOP06 states that byte ranges from -128 to 127. These inclusive endpoints provide 256 values for an 8-bit signed type. Options that include positive 128 exceed the upper endpoint, and those starting at -127 omit the lowest permitted value. Only D preserves both limits correctly.

Why each choice is right or wrong

A - Incorrect: The upper endpoint 128 is one greater than the largest byte value 127.

B - Incorrect: The lower endpoint -127 omits the valid minimum byte value -128.

C - Incorrect: Both endpoints are wrong: -128 is omitted and the invalid positive value 128 is included.

D - Correct: The inclusive limits -128 and 127 exactly match the byte range stated in OOP06.

Primary note proof

Study-note section OOP06

> byte ranges from -128 to 127; short from -32768 to 32767.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean
> and other types).
> • All non-primitive types are reference
> types, so all class types are reference
> types.
> • All integers in Java are signed, Java
> doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 13

Question 42 | Q00444

What is size of integer in Java Programming.

A. 1 Bytes

B. 2 Bytes

C. 4 Bytes

D. 8 Bytes

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer C: 4 bytes (32 bits).

In OOP06's ordered list, int is the 32-bit signed integer type. Converting 32 bits at eight bits per byte gives four bytes. The question concerns Java's fixed int width rather than implementation-dependent C++ sizes. The one-, two- and eight-byte options correspond to different integer widths.

Why each choice is right or wrong

A - Incorrect: One byte corresponds to the 8-bit byte type, not the 32-bit Java int.

B - Incorrect: Two bytes correspond to the 16-bit short width, not the Java int width.

C - Correct: Java int has 32 bits, equivalent to four eight-bit bytes.

D - Incorrect: Eight bytes correspond to the 64-bit long width, rather than the Java int width.

Primary note proof

Study-note section OOP06

> Signed integer widths are 8,16,32,64 bits respectively.

Study-note section OOP-S02

> Java 17 float uses IEEE 754 binary32, and double uses binary64.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean
> and other types).
> • All non-primitive types are reference
> types, so all class types are reference
> types.
> • All integers in Java are signed, Java
> doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 14

Question 43 | Q00445

Which of the following data type(s) can store 64 bit Value.

A. boolean

B. int

C. float

D. long

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer D: long is the 64-bit integer primitive.

OOP06 identifies long as the 64-bit signed integer primitive, following byte, short and int at 8,16 and 32 bits. That directly matches the question's storage-width interpretation. boolean represents logical values, int has only 32 bits, and float is not the stated 64-bit integer storage type.

Why each choice is right or wrong

A - Incorrect: boolean represents logical values rather than providing storage for an arbitrary 64-bit numeric value.

B - Incorrect: The note assigns int a 32-bit width, so it is not the 64-bit integer type.

C - Incorrect: float is a floating numeric type rather than the long integer type explicitly assigned 64 bits.

D - Correct: long is the 64-bit signed integer type identified here's width sequence.

Primary note proof

Study-note section OOP06

> Java has byte, short, int, long, float, double, char and boolean. Signed integer widths are 8,16,32,64 bits respectively.

Study-note section OOP-S02

> Java 17 float uses IEEE 754 binary32, and double uses binary64.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,

and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 15

Question 44 | Q00446

Short data type has a minimum value of _____.

A. -32768

B. -32767

C. 32768

D. 32767

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer A: -32768.

OOP06 gives the Java short range as -32768 through 32767. The question asks for its minimum, so the negative lower endpoint is required. -32767 is one above the minimum; positive 32767 is the maximum, and positive 32768 is outside the stated range entirely.

Why each choice is right or wrong

A - Correct: -32768 is exactly the lower endpoint of the short range given in OOP06.

B - Incorrect: -32767 is representable but is one above the minimum short value.

C - Incorrect: 32768 is above the maximum short value and cannot be its minimum.

D - Incorrect: 32767 is the maximum short value, whereas the question asks for the minimum.

Primary note proof

Study-note section OOP06

byte ranges from -128 to 127; short from -32768 to 32767.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java
(byte, short, int, long, float, double,
char & Boolean)
2. Reference types: Arrays, classes,
and interfaces, String)
• Note: Boolean type has a value true or
false (there is no conversion b/n Boolean
and other types).
• All non-primitive types are reference
types, so all class types are reference
types.
• All integers in Java are signed, Java
doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 16

Question 45 | Q00447

What default value does a Java boolean instance field receive before any explicit initializer or constructor assignment?

A. TRUE

B. false

C. null

D. garbage

Hint

Locate the declaration scope and check default initialization versus definite assignment.

Answer B: false

Java fields receive defined initial values before ordinary explicit initialization. For primitive boolean fields that value is false. true is the other Boolean value, null is a reference value rather than a Boolean value, and Java does not expose an uninitialized field as garbage. The question specifically selects an instance field, so local-variable definite-assignment rules do not change the answer.

Why each choice is right or wrong

A - Incorrect: true is not the default Boolean field value.

B - Correct: false is the default value of a boolean field.

C - Incorrect: null is a reference default and cannot be a primitive boolean value.

D - Incorrect: Java supplies a defined field default rather than a garbage value.

Primary note proof

Study-note section OOP05

> Java fields and array elements receive defaults: numeric zero, char '\u0000', boolean false and references null. A local int t; followed by printing t is a compilation error.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 5 | lines 14-38

> B. Instance variables:
> • Are declared in a class, but
> outside a method, constructor or
> any block.
> • Have default values.
> • are variables which are bound to
> the object itself. Meaning=>
> • Every instance of that class
> (object) has it's own copy of that
> variable
> • Are created when an object is
> created with the use of the
> keyword 'new' and destroyed
> when the object is destroyed.
> • Can be declared in class level
> before or after use.
> • Access modifiers can be given
> for instance variables.
> • Are visible for all methods,
> constructors and block in the
> class.
> E.g. class MyClass {
> float aFloat; }
> NB: By default, a member declared
> within a class is an instance member.

Object Oriented Programming module | PDF page 4 | lines 59-73

> There are three kinds of variables in Java:
> A. Local variables:
> • Local variables are declared in
> methods, constructors, or blocks.
> • has only local scope
> • Access modifiers cannot be used

for local variables.
• They are visible only within the declared method, constructor, or block.
• Have not a default value
• Should be declared and initialized @ z first time.
• Declare before use anywhere in a method or block.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 17

Question 46 | Q00448

What will be the output of the program?

```
class Main {
public static void main(String args[]) {
int t;
System.out.println(t);
}
}
```

A. 0

B. garbage value

C. compiler error

D. runtime error

Hint

Locate the declaration scope and check default initialization versus definite assignment.

Answer C: compiler error

t is a local variable read before assignment. Local variables must be initialized before use and have no default value.

Why each choice is right or wrong

A - Incorrect: Incorrectly applies a field-style default

B - Incorrect: Locals cannot be read as an arbitrary garbage value

C - Correct: Violates the initialization-before-use requirement

D - Incorrect: The violation arises before execution rather than as a runtime failure

Primary note proof

Study-note section OOP05

> Locals occur in methods, constructors or blocks; they must be definitely assigned before use and cannot have public/private/protected modifiers.

Study-note section OOP05

> A local int t; followed by printing t is a compilation error.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 59-73

> There are three kinds of variables in Java:
> A. Local variables:
> • Local variables are declared in
> methods, constructors, or blocks.
> • has only local scope
> • Access modifiers cannot be used
> for local variables.
> • They are visible only within the
> declared method, constructor, or
> block.
> • Have not a default value
> • Should be declared and
> initialized @ z first time.
> • Declare before use anywhere in a
> method or block.

Question sources

Object-Oriented Programming practice questions · Question 18

Question 47 | Q00449

What will be the output of the program?
class Test {
public static void main(String[] args) {
for(int i = 0; 0; i++)
{
System.out.println(""Hello"");
break;
}
}
}

A. Hello

B. Empty Output

C. Compiler error

D. Runtime error

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer C: Compiler error.

The for condition is integer literal 0, whereas OOP06 explicitly says for(...;0;...) is invalid in Java. The doubled quotes around Hello are another source syntax defect. Compilation therefore fails before any loop or print can execute. Treating zero as a false Java condition would incorrectly import C++ numeric truth conversion.

Why each choice is right or wrong

A - Incorrect: No Hello output occurs because the invalid loop condition prevents compilation.

B - Incorrect: Zero is not a legal false condition in Java; this is a compile-time error rather than a valid zero-iteration loop.

C - Correct: The numeric condition and malformed quoted text are compile-time source defects.

D - Incorrect: The program fails translation before execution, so the relevant failure is not a runtime exception.

Primary note proof

Study-note section OOP06

> boolean does not convert to or from numeric truth values; if(0) and for(...;0;...) are invalid.

Study-note section OOP-S01

> An identifier is a programmer-chosen name for an entity such as a class, method or variable. A reserved keyword has a language-defined role and cannot be used as an ordinary identifier. A

literal writes a value directly, such as an integer, floating-point number, quoted character, quoted string or Boolean value.

Study-note section OOP-S03

A while header is while(condition). A basic for header is for(initialization; condition; update). A do-while statement places its condition after the body. An if selects its true branch or, when present, its else branch; only the selected branch executes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Object Oriented Programming module | PDF page 4 | lines 11-20

✓ Reserved keywords: class, extends, implements, for…
✓ Identifiers: programmers given name; grade, gRade…
✓ Literals (Constants values): 55, 60.78, “Hi”, „M‟… Java language specifies five major types of literals => integer, floating, character, string & Boolean literals

Question sources

Object-Oriented Programming practice questions · Question 19

Question 48 | Q00450

What will be the output of the program?

```
class mainclass {
public static void main(String args[])
{
boolean var1 = true;
boolean var2 = false;
if (var1)
System.out.println(var1);
else
System.out.println(var2);
}
}
```

A. 0

B. 1

C. true

D. FALSE

Hint

Respect operator grouping and operand types, then trace each operation in evaluation order.

Answer C: true

var1 is initialized to true, so the if body executes and prints var1. The else body is skipped, so var2 does not appear. Java's Boolean output is the lowercase word true, followed by the newline from println. It does not use numeric 1 or uppercase TRUE. Correcting C's case makes it the exact visible output rather than an approximate description.

Why each choice is right or wrong

A - Incorrect: The chosen true Boolean value does not print as numeric 0.

B - Incorrect: Java Boolean output is not encoded as numeric 1.

C - Correct: The true branch prints the lowercase word true.

D - Incorrect: The false value is not printed because the else branch is skipped.

Primary note proof

Study-note section OOP23

> Printing Boolean true gives lowercase true, not 1 or uppercase TRUE.

Study-note section OOP-S03

> A while header is while(condition). A basic for header is for(initialization; condition; update). A do-while statement places its condition after the body. An if selects its true branch or, when present, its else branch; only the selected branch executes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Object Oriented Programming module | PDF page 7 | lines 48-54

2. Declaration statements: declares/ Introduces a very first variables E.g. double c = 8.4;
3. Control flow statements: regulates the order of statements execution
Selection: the if() and switch() statements

Question sources

Object-Oriented Programming practice questions · Question 20

Question 49 | Q00451

What is the numerical value computed by the final expression, rounded to eight digits after the decimal point? This asks for the rounded value, not the exact unformatted console string.

```
class average {
public static void main(String args[])
{
double num[] = {5.5, 10.1, 11, 12.8, 56.9, 2.5};
double result;
result = 0;
for (int i = 0; i<6; ++i)
result = result + num[i];
System.out.print(result/6);
}
}
```

A. 16.34

B. 16.56666664

C. 16.46666667

D. 16.76666667

Hint

Trace every element or API argument and distinguish copying values from aliasing a reference.

Answer C: 16.46666667

The six listed numbers sum to 98.8 mathematically. Dividing by six gives approximately 16.4666666667, which rounds to 16.46666667 at eight decimal places. Small binary-double representation differences do not affect that rounding here. The original unformatted print call does not impose eight-place formatting, so the revised question explicitly requests a rounded numerical value rather than claiming this is its exact console string.

Why each choice is right or wrong

A - Incorrect: 16.34 is not the sum divided by six at the requested precision.

B - Incorrect: 16.56666664 differs from the computed mean before rounding.

C - Correct: 16.46666667 is the mean rounded to eight decimal places.

D - Incorrect: 16.76666667 does not match the computed mean.

Primary note proof

Study-note section OOP27

> For int[] a=new int[4], setting a[i]=i*3 and accumulating yields 0+3+6+9=18. To average an array, sum every element and divide by its length using floating arithmetic; handle zero length separately. Floating representations and printing can differ from a rounded mathematical result. For the six-value example, the mathematical average is 98.8/6, approximately 16.46666667; the exact printed string also depends on floating representation and formatting.

Study-note section OOP26

Valid indices are 0..a.length-1; an invalid access throws ArrayIndexOutOfBoundsException.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 21

Question 50 | Q00452

What will be the output of the program?
class increment {
public static void main(String args[])
{
int g = 4;
System.out.print(++g * 8);
}
}

A. 32

B. 36

C. 40

D. 48

Hint

Track the old and updated values separately and distinguish a character operand from a string.

Answer C: 40.

The prefix increment changes g from 4 to 5 before its value is used in multiplication. The expression therefore computes 5*8=40 and prints that integer. This differs from postfix increment, which would supply the old value to the multiplication. Only one increment occurs in this expression.

Why each choice is right or wrong

A - Incorrect: 32 would multiply the old value 4 by 8, which does not respect prefix increment.

B - Incorrect: 36 is not the product of either the old value 4 or updated value 5 with 8.

C - Correct: Prefix increment supplies 5, so the printed product is 5 times 8, or 40.

D - Incorrect: 48 requires g to become 6, but the single increment changes it only from 4 to 5.

Primary note proof

Study-note section OOP24

> Starting x=5, printing x++ then ++x gives 5 then 7.

Study-note section OOP23

> Java evaluates operands left to right while respecting precedence.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that
> take one or more arguments

(operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Question sources

Object-Oriented Programming practice questions · Question 22

Question 51 | Q00453

What will this program print when run on Java 17?

```
class area {
public static void main(String args[])
{
double r, pi, a;
r = 9.8;
pi = 3.14;
a = pi * r * r;
System.out.println(a);
}
}
```

A. 301.5656

B. 301

C. 301.56

D. 301.57

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer A: 301.5656

Java groups pi * r * r as (pi * r) * r and rounds each double multiplication using binary64 arithmetic. The resulting value has hexadecimal representation 0x1.2d90cb295e9e2p+8. Java 17's standard conversion prints that value as 301.5656, followed by println's line terminator. The supplementary Oracle rules establish the precise arithmetic and text conversion; no separate decimal-place formatting is applied.

Why each choice is right or wrong

A - Correct: 301.5656 is the standard Java 17 printed representation of the computed double.

B - Incorrect: There is no integer conversion that discards the fractional part.

C - Incorrect: The program does not truncate or format the result to two decimal places.

D - Incorrect: The program does not round its displayed output to two decimal places.

Primary note proof

Study-note section OOP-S02

> Java 17 float uses IEEE 754 binary32, and double uses binary64. Successive multiplications group from left to right. Each floating-point operation rounds to its representable result, using nearest-value rounding with ties to even; regrouping a product can change its result.

Study-note section OOP-S02

> PrintStream.print(double) converts the stored value through String.valueOf and Double.toString. For finite nonzero magnitudes from 0.001 inclusive to 10000000 exclusive, the standard decimal form uses at least one fractional digit and only the further digits needed to distinguish the value from adjacent doubles. println(double) also adds a line terminator. This is not fixed-decimal-place formatting.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 23

Question 52 | Q00454

What will be the output of the program?

```
class increment {
public static void main(String args[])
{
int g = 6;
System.out.print(--g * 8);
}
}
```

A. 48

B. 40

C. 56

D. 44

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer B: 40.

The prefix decrement reduces g from 6 to 5 before multiplication uses it. The resulting integer expression is 5*8, which equals 40. The updated value is used immediately; it is not the old 6 and no later increment occurs. Thus the integer printed is 40.

Why each choice is right or wrong

A - Incorrect: 48 uses the old value 6, which would not reflect the prefix decrement before multiplication.

B - Correct: Prefix decrement supplies 5 to the multiplication, producing 5*8=40.

C - Incorrect: 56 uses 7, but decrement reduces the initial 6 instead of increasing it.

D - Incorrect: 44 is not 5 multiplied by 8 and does not follow the stated prefix update.

Primary note proof

Study-note section OOP-S03

> Prefix ++ and -- update a variable before supplying its value; postfix forms supply the old value before the update. Multiplication, division and remainder bind before addition and subtraction; same-level binary arithmetic groups left to right. Integer division truncates toward zero.

Study-note section OOP23

> Java evaluates operands left to right while respecting precedence.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Question sources

Object-Oriented Programming practice questions · Question 24

Question 53 | Q00456

To declare more than one variable of the specified type, we can use a _____ list.

A. colon-separated

B. bracket-separated

C. comma-separated

D. None of the above

Hint

Locate the declaration scope and check default initialization versus definite assignment.

Answer C: A comma-separated declarator list.

OOP05 illustrates a declaration with int a=10,b=10, separating the declarators with a comma. The shared type applies to each name in that declaration. Colons and brackets serve other syntactic purposes and do not separate ordinary variable declarators in the example. The comma-separated option states the required list form.

Why each choice is right or wrong

A - Incorrect: A colon does not separate the ordinary variable declarators shown in the note's shared-type declaration.

B - Incorrect: Brackets describe array types or indexing rather than separating names in a declarator list.

C - Correct: The example int a=10,b=10 uses a comma between variable declarators sharing the type int.

D - Incorrect: Comma-separated is an offered and supported list form, so none of the above is false.

Primary note proof

Study-note section OOP05

> Initializers such as int a=10,b=10; assign initial values.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 11-20

> ✓ Reserved keywords: class, extends, implements, for…
> ✓ Identifiers: programmers given name; grade, gRade…
> ✓ Literals (Constants values): 55, 60.78, "Hi", „M"… Java language specifies five major types of literals => integer,

floating, character, string &
Boolean literals

Question sources

Object-Oriented Programming practice questions · Question 26

Question 54 | Q00457

In the introductory grouping by declaration scope and class-versus-instance ownership, how many categories of Java variables are distinguished? Include parameter and block-local contexts within the local category.

A. 2

B. 3

C. 4

D. 5

Hint

Locate the declaration scope and check default initialization versus definite assignment.

Answer B: 3

The introductory grouping distinguishes local variables, instance fields and class/static fields. Locals belong to method, constructor or block execution; instance fields belong to individual objects; static fields belong to the class. This gives three categories. The question uses that teaching grouping rather than counting every variable category distinguished by the full language specification.

Why each choice is right or wrong

A - Incorrect: Two categories cannot account for all three of local, instance and class/static variables.

B - Correct: The specified introductory grouping contains local, instance and class/static categories: three.

C - Incorrect: Four exceeds the three categories in the specified grouping.

D - Incorrect: Five exceeds the three categories in the specified grouping.

Primary note proof

Study-note section OOP05

The introductory three-way grouping is local, instance and class/static variables.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 59-73

There are three kinds of variables in Java:
A. Local variables:
• Local variables are declared in methods, constructors, or blocks.
• has only local scope
• Access modifiers cannot be used for local variables.
• They are visible only within the declared method, constructor, or block.
• Have not a default value
• Should be declared and initialized @ z first time.
• Declare before use anywhere in a method or block.

Object Oriented Programming module | PDF page 5 | lines 14-38

B. Instance variables:
• Are declared in a class, but outside a method, constructor or any block.
• Have default values.
• are variables which are bound to the object itself. Meaning=>
• Every instance of that class (object) has it's own copy of that variable
• Are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
• Can be declared in class level before or after use.
• Access modifiers can be given for instance variables.
• Are visible for all methods, constructors and block in the class.
E.g. class MyClass {
float aFloat; }
NB: By default, a member declared within a class is an instance member.

Object Oriented Programming module | PDF page 5 | lines 39-61

C. Class/Static variables
• Are declared with the static

keyword in a class, but outside a method, constructor or a block.
• There would only be one copy of each class variable per class, regardless of how many objects are created from it.
• Are rarely used other than being declared as constants. (public static final)
• Are stored in the static memory.
• Static variables are created when the program starts and destroyed when the program stops.
• Visibility is similar to instance variables. However, most static variables are declared public
• Default values are same as instance variables.
• Static variables can be accessed by calling with the class name ClassName.VariableName.

Question sources

Object-Oriented Programming practice questions · Question 27

Question 55 | Q00458

Local variables are declared in?

A. methods

B. constructors

C. blocks

D. All of the above

Hint

Locate the declaration scope and check default initialization versus definite assignment.

Answer D: All of the above

The local-variable definition explicitly includes methods, constructors and blocks.

Why each choice is right or wrong

A - Incorrect: One of the three listed contexts

B - Incorrect: One of the three listed contexts

C - Incorrect: One of the three listed contexts

D - Correct: Combines all three permitted contexts

Primary note proof

Study-note section OOP05

> Locals occur in methods, constructors or blocks; they must be definitely assigned before use and cannot have public/private/protected modifiers.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 59-73

> There are three kinds of variables in Java:
> A. Local variables:
> • Local variables are declared in methods, constructors, or blocks.
> • has only local scope
> • Access modifiers cannot be used for local variables.
> • They are visible only within the declared method, constructor, or block.
> • Have not a default value
> • Should be declared and initialized @ z first time.
> • Declare before use anywhere in a method or block.

Question sources

Object-Oriented Programming practice questions · Question 28

Question 56 | Q00459

Which qualifier is preferred for accessing an accessible Java static field to make its class-level ownership clear?

A. Object name

B. Class name

C. Function name

D. Can not say

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer B: Class name

A static field belongs to the loaded class and is shared by its instances. ClassName.field makes that ownership explicit and is the recommended qualification. An object reference can sometimes qualify a static member, but it obscures the class-level nature and is not the preferred form asked for. Function names do not supply the owning type, and the style rule gives a definite answer.

Why each choice is right or wrong

A - Incorrect: Object qualification can be legal, but it is not the preferred class-ownership notation.

B - Correct: Class qualification is the preferred notation for a static field.

C - Incorrect: A function name does not identify the field's owning type as a qualifier.

D - Incorrect: The note provides a definite preferred class-qualified form.

Primary note proof

Study-note section OOP07

> static gives one field per loaded class, shared by its instances. If count starts at zero and every constructor increments it, three creations leave count=3. Prefer ClassName.count and ClassName.utility().

Study-note section OOP16

> With Animal a=new Dog(), a.staticMethod() resolves using Animal's compile-time type. Prefer Animal.staticMethod() for clarity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 5 | lines 39-61

> C. Class/Static variables
> • Are declared with the static keyword in a class, but outside a method, constructor or a block.
> • There would only be one copy of each class variable per class, regardless of how many objects are created from it.
> • Are rarely used other than being declared as constants. (public static final)
> • Are stored in the static memory.
> • Static variables are created when the program starts and destroyed when the program stops.
> • Visibility is similar to instance variables. However, most static

variables are declared public
• Default values are same as instance variables.
• Static variables can be accessed by calling with the class name ClassName.VariableName.

Object Oriented Programming module | PDF page 14 | lines 2-19

Static methods and Final Variables
• A static method can access only static data.
• A static method can call only other static data (Static methods & variables), but can not call a non-static data from it.
• Static method is belongs to the class and not to the object.
• A static method can be accessed directly by the class name and doesn‟t need any object.
• i.e ClassName.methodName(args)
• A static method cannot refer to "this" or "super" keywords in anyway
• But can be overloaded since they are resolved using static binding by compiler at compile time.

Question sources

Object-Oriented Programming practice questions · Question 29

Question 57 | Q00460

Which of the following is an Example of variable initialization?

A. int a, b, c;

B. int a = 10, b = 10;

C. int 10 = a;

D. None of the above

Hint

Locate the declaration scope and check default initialization versus definite assignment.

Answer B: int a = 10, b = 10;

The declaration double c = 8.4 illustrates assignment of an initial value at declaration; B follows the same type-name-equals-value pattern for a and b.

Why each choice is right or wrong

A - Incorrect: Declares variables without initial values

B - Correct: Assigns initial values 10 to named variables

C - Incorrect: Places a literal where the declared name belongs

D - Incorrect: B demonstrates initialization

Primary note proof

Study-note section OOP05

> Initializers such as int a=10,b=10; assign initial values.

Study-note section OOP-S01

> An identifier is a programmer-chosen name for an entity such as a class, method or variable. A reserved keyword has a language-defined role and cannot be used as an ordinary identifier. A literal writes a value directly, such as an integer, floating-point number, quoted character, quoted string or Boolean value.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 7 | lines 48-54

> 2. Declaration statements: declares/
> Introduces a very first variables E.g.
> double c = 8.4;
> 3. Control flow statements: regulates
> the order of statements execution
> Selection: the if() and switch()
> statements

Question sources

Object-Oriented Programming practice questions · Question 30

Question 58 | Q00461

Access modifiers cannot be used for local variables.

A. Yes

B. No

C. Can be yes or no

D. Cannot say

Hint

Locate the declaration scope and check default initialization versus definite assignment.

Answer A: Yes

The local-variable section explicitly says access modifiers cannot be used for local variables.

Why each choice is right or wrong

A - Correct: Agrees with the rule

B - Incorrect: Contradicts the rule

C - Incorrect: The stated prohibition is not conditional

D - Incorrect: The rule gives a definite answer

Primary note proof

Study-note section OOP05

> Locals occur in methods, constructors or blocks; they must be definitely assigned before use and cannot have public/private/protected modifiers.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 59-73

> There are three kinds of variables in Java:
> A. Local variables:
> • Local variables are declared in
> methods, constructors, or blocks.
> • has only local scope
> • Access modifiers cannot be used
> for local variables.
> • They are visible only within the
> declared method, constructor, or
> block.
> • Have not a default value
> • Should be declared and
> initialized @ z first time.
> • Declare before use anywhere in a
> method or block.

Question sources

Object-Oriented Programming practice questions · Question 31

Question 59 | Q00462

Which of the following can be operands of arithmetic operators?

A. Characters

B. Boolean

C. Numeric

D. Both Numeric & Characters

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer D: Both numeric values and characters can participate in arithmetic.

OOP06 states that arithmetic promotes char, byte and short to int in many expressions. Characters are therefore allowed alongside numeric operands. boolean does not convert to numeric truth values and is not an arithmetic operand. D gives the complete supported pair; A and C each mention only part of that operand domain.

Why each choice is right or wrong

A - Incorrect: Characters are permitted through promotion, but this option alone omits the also-permitted numeric operand category.

B - Incorrect: Java boolean does not convert to numeric values for arithmetic, as OOP06 explicitly states.

C - Incorrect: Numeric operands are permitted, but this option alone omits the also-permitted character category.

D - Correct: Both numeric and char operands are permitted; chars undergo numeric promotion for arithmetic.

Primary note proof

Study-note section OOP06

> Arithmetic promotes char, byte and short to int in many expressions. boolean does not convert to or from numeric truth values; if(0) and for(...;0;...) are invalid.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary),

or three operands (Ternary).
Java has one ternary operator,
?:, which is a short-hand if-
else statement. E.g. c=x?w:s;
Operators can be Arithmetic,
Logical, assignment,
Relational, conditional,
Increment,

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java
(byte, short, int, long, float, double,
char & Boolean)
2. Reference types: Arrays, classes,
and interfaces, String)
• Note: Boolean type has a value true or
false (there is no conversion b/n Boolean
and other types).
• All non-primitive types are reference
types, so all class types are reference
types.
• All integers in Java are signed, Java
doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 32

Question 60 | Q00463

Modulus operator, %, can be applied to which of these?

A. Both Integers and floating - point numbers

B. Integers

C. Floating - point numbers

D. None of the mentioned

Hint

Respect operator grouping and operand types, then trace each operation in evaluation order.

Answer A: Both integers and floating-point numbers in Java.

OOP23 explicitly contrasts Java with C++: Java % accepts floating-point operands as well as integers. This course's Java context therefore makes A the complete domain among the offered choices. B and C each name only one supported category. Importing C++'s integral-only restriction would give the wrong answer for Java.

Why each choice is right or wrong

A - Correct: Both integer and floating-point operands are accepted by Java's remainder operator according to OOP23.

B - Incorrect: Integers are allowed, but this option alone omits Java's also-supported floating-point operands.

C - Incorrect: Floating-point values are allowed, but this option alone omits the also-supported integer operands.

D - Incorrect: The note explicitly supports both supplied operand categories, so none of the mentioned is false.

Primary note proof

Study-note section OOP23

> Unlike C++, Java % also accepts floating-point operands.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Question sources

Object-Oriented Programming practice questions · Question 33

Question 61 | Q00465

Evaluate the value of the expression?
6 - 2 + 10 % 4 + 7

A. 14

B. 12

C. 13

D. 10

Hint

Respect operator grouping and operand types, then trace each operation in evaluation order.

Answer C: 13.

The remainder operation is evaluated first: 10%4 is 2. The remaining additions and subtraction group left to right, yielding 6-2+2+7 = 4+2+7 = 13. OOP23 includes the same expression and result. No floating conversion or increment operation appears in this calculation.

Why each choice is right or wrong

A - Incorrect: 14 does not follow the remainder 10%4=2 and the resulting sum 6-2+2+7.

B - Incorrect: 12 is one below the computed total and does not respect the stated remainder result.

C - Correct: Evaluating 10%4 first gives 2; then 6-2+2+7 equals 13.

D - Incorrect: 10 omits or miscalculates part of the expression; the full integer evaluation gives 13.

Primary note proof

Study-note section OOP23

> Java evaluates operands left to right while respecting precedence.

Study-note section OOP23

> 6-2+10%4+7 is 13.

Study-note section OOP-S03

> Prefix ++ and -- update a variable before supplying its value; postfix forms supply the old value before the update. Multiplication, division and remainder bind before addition and subtraction; same-level binary arithmetic groups left to right. Integer division truncates toward zero.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that
> take one or more arguments
> (operands) and operates on
> them to produce a result. An

operator performs a function
on one (Unary), two (Binary),
or three operands (Ternary).
Java has one ternary operator,
?:, which is a short-hand if-
else statement. E.g. c=x?w:s;
Operators can be Arithmetic,
Logical, assignment,
Relational, conditional,
Increment,

Question sources

Object-Oriented Programming practice questions · Question 35

Question 62 | Q00466

What is the output of this program?
class Main {
public static void main(String args[])
{
double var1 = 2 + 4;
double var2 = var1 / 4;
int var3 = 2 + 4;
int var4 = var3 / 4;
System.out.print(var2 + " " + var4);
}
}

A. 0 1

B. 1 1

C. 1.5 1

D. 1.5 1.0

Hint

Respect operator grouping and operand types, then trace each operation in evaluation order.

Answer C: 1.5 1.

Both additions initially produce 6. Assigning one to double makes var1=6.0, so var1/4 gives 1.5. var3 remains int, so var3/4 truncates to integer 1. The string containing a space then concatenates their representations, producing 1.5 1 rather than giving the integer result a decimal suffix.

Why each choice is right or wrong

A - Incorrect: The double division of 6.0 by 4 yields 1.5, not zero.

B - Incorrect: The first quotient is floating point and retains .5 rather than becoming integer 1.

C - Correct: The double quotient prints 1.5, followed by the space and integer quotient 1.

D - Incorrect: var4 is int, so its string representation is 1 rather than 1.0.

Primary note proof

Study-note section OOP23

> Integer 6/4 is 1; double 6.0/4 is 1.5.

Study-note section OOP24

> Once a String operand is encountered, + concatenates in left-to-right groups: 1+2+"x" gives "3x", while "x"+1+2 gives "x12".

Study-note section OOP-S03

> A while header is while(condition). A basic for header is for(initialization; condition; update). A do-while statement places its condition after the body. An if selects its true branch or, when present, its else branch; only the selected branch executes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that
> take one or more arguments
> (operands) and operates on
> them to produce a result. An
> operator performs a function
> on one (Unary), two (Binary),
> or three operands (Ternary).
> Java has one ternary operator,
> ?:, which is a short-hand if-
> else statement. E.g. c=x?w:s;
> Operators can be Arithmetic,
> Logical, assignment,
> Relational, conditional,
> Increment,

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean
> and other types).
> • All non-primitive types are reference

types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 36

Question 63 | Q00467

What will be the output of the program?

```
class Bitwise
{
public static void main(String [] args)
{
int x = 11 & 9;
int y = x ^ 3;
System.out.println( y | 12 );
}
}
```

A. 7

B. 0

C. 14

D. 8

Hint

Respect operator grouping and operand types, then trace each operation in evaluation order.

Answer C: 14.

The AND produces 11&9 = 1011&1001 = 1001, or 9. XOR with 3 gives 1001^0011 = 1010, or 10. OR with 12 then gives 1010|1100 = 1110, or 14. OOP23 explicitly provides this chain; these are integer bitwise operations, not Boolean short-circuit operations.

Why each choice is right or wrong

A - Incorrect: Seven has binary 0111, whereas the final OR produces 1110.

B - Incorrect: The final OR includes the set bits of 12, so it cannot produce zero.

C - Correct: The chain produces x=9, y=10 and final y|12=14.

D - Incorrect: Eight omits bits set by the final OR of binary 1010 and 1100.

Primary note proof

Study-note section OOP23

For bitwise values, write binary columns: 11&9=9; 9^3=10; 10|12=14.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Question sources

Object-Oriented Programming practice questions · Question 37

Question 64 | Q00468

What is the output of relational operators?

A. Integer

B. Boolean

C. Characters

D. Double

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer B: Boolean.

Relational expressions test a relation rather than return one of their numeric operands. In Java that truth result is Boolean, consistent with OOP06's prohibition on treating boolean as numeric truth values and OOP23's Boolean output discussion. Integer, character and double values can be compared in suitable expressions, but are not the comparison result type.

Why each choice is right or wrong

A - Incorrect: Java comparison results are logical values rather than integer truth codes such as zero and one.

B - Correct: A relational test yields the Boolean truth value true or false.

C - Incorrect: Characters can be compared, but a comparison does not return a character result.

D - Incorrect: Double operands may participate in comparisons, but their relation yields a Boolean rather than a double.

Primary note proof

Study-note section OOP-S03

> Java relational comparisons (<, <=, >, >=) and equality tests (==, !=) produce primitive boolean truth values. Their result is not an integer truth code or a value of the numeric operand type.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that
> take one or more arguments
> (operands) and operates on
> them to produce a result. An
> operator performs a function
> on one (Unary), two (Binary),
> or three operands (Ternary).
> Java has one ternary operator,
> ?:, which is a short-hand if-
> else statement. E.g. c=x?w:s;
> Operators can be Arithmetic,
> Logical, assignment,
> Relational, conditional,
> Increment,

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean
> and other types).
> • All non-primitive types are reference
> types, so all class types are reference
> types.

• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 38

Question 65 | Q00469

Which of these is returned by "greater than", "less than" and "equal to" operators?

A. Integers

B. Floating - point numbers

C. Boolean

D. None of the mentioned

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer C: Boolean true or false.

Greater-than, less-than and equality operators express truth tests. Java represents their results as Boolean values, not integers or floating-point numbers. OOP06 distinguishes boolean from numeric values, and OOP23 describes printing Boolean true as lowercase true. The operand type therefore does not become the result type of the test.

Why each choice is right or wrong

A - Incorrect: Java uses Boolean truth values for comparisons rather than returning integer zero or one.

B - Incorrect: Comparing floating operands still yields a truth value, not a floating-point numerical result.

C - Correct: Greater-than, less-than and equality comparisons return Boolean true or false.

D - Incorrect: Boolean is explicitly offered and is the appropriate logical result type, so none is false.

Primary note proof

Study-note section OOP-S03

Java relational comparisons (<, <=, >, >=) and equality tests (==, !=) produce primitive boolean truth values. Their result is not an integer truth code or a value of the numeric operand type.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 39

Question 66 | Q00471

In Java, what type must expression1 have in the conditional expression expression1 ? expression2 : expression3?

A. Integer

B. Floating - point numbers

C. Boolean

D. None of the mentioned

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer C: Java requires boolean or Boolean for the first operand of ?: .

Java's conditional operator requires its first operand to have primitive boolean or boxed Boolean type. That condition chooses which of the other two expressions is evaluated; the unselected alternative is not evaluated. Integer and floating-point values are not accepted as numeric truth codes. The Boolean option therefore names the required logical category, and None is false because that category is offered.

Why each choice is right or wrong

A - Incorrect: Java integer values do not satisfy the JLS first-operand requirement of boolean or Boolean.

B - Incorrect: Floating-point values do not satisfy Java's Boolean conditional-operand requirement.

C - Correct: Boolean names the required logical category; Java accepts primitive boolean or boxed Boolean.

D - Incorrect: The Boolean category is offered and satisfies the JLS requirement, so none is false.

Primary note proof

Study-note section OOP-S03

> The conditional expression condition ? whenTrue : whenFalse requires a boolean or Boolean condition. It evaluates only the selected alternative. An integer or floating-point first operand is a compile-time error.

Study-note section OOP06

> boolean does not convert to or from numeric truth values; if(0) and for(...;0;...) are invalid.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic,

Logical, assignment,
Relational, conditional,
Increment,

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java
(byte, short, int, long, float, double,
char & Boolean)
2. Reference types: Arrays, classes,
and interfaces, String)
• Note: Boolean type has a value true or
false (there is no conversion b/n Boolean
and other types).
• All non-primitive types are reference
types, so all class types are reference
types.
• All integers in Java are signed, Java
doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 41

Question 67 | Q00472

What is the value stored in x in following lines of code?
int x, y, z;
x = 0;
y = 1;
x = y = z = 8;

A. 0

B. 1

C. 9

D. 8

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer D: x becomes 8.

The last statement assigns 8 to z, then that same assigned value to y and then x. The earlier x=0 and y=1 values are overwritten, and no addition takes place. This follows assignment's value propagation rather than the left-to-right operand arithmetic discussed in OOP23. The final stored value of x is 8.

Why each choice is right or wrong

A - Incorrect: Zero is x's earlier value; the final chained assignment overwrites it with 8.

B - Incorrect: One is y's earlier value, which is also overwritten before x receives the chain's result.

C - Incorrect: The statement performs assignments, not addition of the earlier y value to 8.

D - Correct: The chain assigns 8 to z, then y and x, leaving x equal to 8.

Primary note proof

Study-note section OOP-S03

> Assignments group right to left: a=b=c assigns c to b, then the resulting value to a. A later assignment overwrites an earlier value. Compound assignments update the left variable using the indicated operation.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Question sources

Object-Oriented Programming practice questions · Question 42

Question 68 | Q00473

What is the output of this program?

```
class Main
{
public static void main(String args[])
{
int x = 8;
System.out.println(++x * 3 + " " + x);
}
}
```

A. 24 8

B. 24 9

C. 27 8

D. 27 9

Hint

Track the old and updated values separately and distinguish a character operand from a string.

Answer D: 27 9.

Prefix ++x changes x from 8 to 9 before multiplication, producing 27. Concatenating the string containing a space switches the remaining + operations to text concatenation. The later x still holds 9 because no further update occurs. Therefore the single println emits 27 9, followed by its newline.

Why each choice is right or wrong

A - Incorrect: Both values wrongly use the original x=8 despite the prefix increment before multiplication.

B - Incorrect: The final x is 9, but the product must also use updated 9 and become 27.

C - Incorrect: The product is 27, but x remains 9 rather than reverting to its original 8.

D - Correct: The prefix update gives 9, its product is 27, and concatenation then appends space and 9.

Primary note proof

Study-note section OOP24

> Starting x=5, printing x++ then ++x gives 5 then 7.

Study-note section OOP24

> Once a String operand is encountered, + concatenates in left-to-right groups: 1+2+"x" gives "3x", while "x"+1+2 gives "x12".

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Question sources

Object-Oriented Programming practice questions · Question 43

Question 69 | Q00474

_____ are a sequence of characters.

A. Character

B. Strings

C. Integer

D. Classes

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer B: Strings

The String Class description explicitly equates strings with a sequence of chars.

Why each choice is right or wrong

A - Incorrect: A character is one element, not the named sequence type

B - Correct: The sequence-of-characters type

C - Incorrect: Integer represents a different primitive category

D - Incorrect: Class is the general blueprint concept rather than this sequence type

Primary note proof

Study-note section OOP-S04

> A String represents a sequence of UTF-16 code units. A single char represents one code unit rather than an entire String. String objects are immutable; changing a reference to another String does not modify the original text value.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 14-16

> 5. String Class:- Immutable
> Class created to manipulate
> strings(Sequence of chars).

Question sources

Object-Oriented Programming practice questions · Question 44

Question 70 | Q00475

The String class is?

A. mutable

B. immutable

C. Both A and B

D. None of the above

Hint

Separate identity, content equality, immutable values and references that may be reassigned.

Answer B: immutable

String is immutable, unlike mutable classes whose state can change.

Why each choice is right or wrong

A - Incorrect: Contradicts the stated immutable classification

B - Correct: Matches the stated classification

C - Incorrect: The two opposite classifications do not both describe String here

D - Incorrect: B is directly supported

Primary note proof

Study-note section OOP22

> Java String objects are immutable. text += " World" assigns a reference to a new concatenated result and can print Hello World without mutating the original String object. final class String prevents subclassing; final alone does not make arbitrary objects immutable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 14-16

> 5. String Class:- Immutable
> Class created to manipulate
> strings(Sequence of chars).

Question sources

Object-Oriented Programming practice questions · Question 45

Question 71 | Q00476

Methods used to obtain information about an object are known as?

A. string methods

B. class methods

C. object method

D. accessor methods

Hint

Separate parameters from arguments and operations that read state from operations that change it.

Answer D: Accessor methods.

OOP11 says an accessor or getter returns information, whereas a mutator or setter changes state. Methods used to obtain information about an object therefore have the specific name accessor methods. String, class and object describe other categories or associations and do not identify this information-reading responsibility.

Why each choice is right or wrong

A - Incorrect: String methods concern a particular class; the requested information-reading role applies to objects generally.

B - Incorrect: Class methods identify a class/static association, rather than the specific purpose of returning object information.

C - Incorrect: Object method is too general; object methods can either read or change state.

D - Correct: Accessor methods specifically return information about object state, matching the definition in OOP11.

Primary note proof

Study-note section OOP11

> An accessor/getter returns information; a mutator/setter changes state.

Study-note section OOP07

> Prefer ClassName.count and ClassName.utility().

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 10 | lines 62-72

> Methods
> • Are members of classes and do operations.
> • Allows different classes to communicate each other.

• E.g. type name(parameter-
list)
{ // body of method
//....
return value; // if
type is not void

Object Oriented Programming module | PDF page 10 | lines 15-26

• Every Dog object will contain its own copies of the instance variables breed, age, and color.
• To access these variables, you will use the dot (.) operator.
• The dot operator links the name of the object with the name of an instance variable.
• For example, to assign the age variable of myDog the value 10, you would use the following statement:
myDog.age = 10; //This is called

Question sources

Object-Oriented Programming practice questions · Question 46

Question 72 | Q00477

Which method returns the character at the specified index?

A. int compareTo(Object o)

B. int compareTo(String anotherString)

C. char charAt(int index)

D. int compareToIgnoreCase(String str)

Hint

Separate identity, content equality, immutable values and references that may be reassigned.

Answer C: char charAt(int index).

OOP22 explicitly says charAt(index) returns a char. C gives that operation's parameter and result type: an integer position selects a single character. The compareTo alternatives return int comparison results rather than the character at a position, so they do not satisfy the requested retrieval behavior.

Why each choice is right or wrong

A - Incorrect: This comparison signature takes an object rather than an index and returns int instead of a character.

B - Incorrect: This comparison signature takes another String and returns int, rather than retrieving a character by index.

C - Correct: charAt takes an integer index and returns the char at that position, matching OOP22.

D - Incorrect: This comparison signature takes a String, not an integer index, and does not return the indexed char.

Primary note proof

Study-note section OOP22

> charAt(index) returns a char; length() counts UTF-16 units; substring(begin,end) excludes end.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 14-16

> 5. String Class:- Immutable
> Class created to manipulate
> strings(Sequence of chars).

Question sources

Object-Oriented Programming practice questions · Question 47

Question 73 | Q00478

Which of these methods of String class is used to obtain character at specified index?

A. char()

B. Charat()

C. charat()

D. charAt()

Hint

Separate identity, content equality, immutable values and references that may be reassigned.

Answer D: charAt (with an index argument).

The operation named in OOP22 is charAt(index), with a lowercase c and uppercase A. D alone preserves that exact Java method name. The options abbreviate away the index argument; an actual call needs it. Java's case sensitivity prevents Charat or charat from naming the same standard operation.

Why each choice is right or wrong

A - Incorrect: char is a primitive type keyword, not the indexed String character-access method name.

B - Incorrect: Charat has the wrong capitalization and does not name the charAt operation in OOP22.

C - Incorrect: charat lacks the uppercase A required by the case-sensitive standard method name.

D - Correct: charAt is the exact method name; a real invocation supplies an index inside the parentheses.

Primary note proof

Study-note section OOP22

> charAt(index) returns a char; length() counts UTF-16 units; substring(begin,end) excludes end.

Study-note section OOP28

> Java identifiers are case sensitive; $ is allowed but @ is not an identifier character.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 14-16

> 5. String Class:- Immutable
> Class created to manipulate
> strings(Sequence of chars).

Question sources

Object-Oriented Programming practice questions · Question 48

Question 74 | Q00480

Which Java loop form pre-tests its body condition and writes that condition alone in parentheses immediately after its loop keyword, rather than using a three-clause header or a post-test?

A. for

B. do-while

C. while

D. None of the above

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer C: while

A while loop tests its condition before every possible execution and has a condition-only header. A for loop is also a pre-test loop, but its ordinary header has initialization, condition and update clauses. A do-while loop tests after its body and therefore runs at least once. The added header and timing distinctions make while the unique match rather than relying on repetition alone.

Why each choice is right or wrong

A - Incorrect: A basic for uses the three-clause header excluded by the stem.

B - Incorrect: A do-while post-tests rather than pre-testing its first iteration.

C - Correct: while has the condition-only pre-test form described.

D - Incorrect: The while form satisfies the description, so none is incorrect.

Primary note proof

Study-note section OOP-S03

> A while header is while(condition). A basic for header is for(initialization; condition; update). A do-while statement places its condition after the body. An if selects its true branch or, when present, its else branch; only the selected branch executes.

Study-note section OOP25

> while and for pre-test and may execute zero times; do-while post-tests and executes once.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 7 | lines 26-41

> Iteration: The for(), while() and
> do...while() statements
> 4. Jumping Statements: to jump out
> of loops and to control other areas of

program java uses break and
continue statements.
The break statement: used to exit a
loop
The Continue Statement: used to
force program to go back to the top
of a loop
Labels: Java does not include a goto
statement.
Instead of goto, Java allows you to
combine break and continue with a
label.

Question sources

Object-Oriented Programming practice questions · Question 50

Question 75 | Q00481

Which loop guaranteed to execute at least one time?

A. for

B. do-while

C. while

D. All of the above

Hint

Trace condition tests, control transfers and printing as separate steps.

Answer B: do-while.

OOP25 states that do-while post-tests and executes once, whereas for and while pre-test and may execute zero times. The mandatory first body execution is therefore specific to do-while among these forms. This guarantee concerns entering its body when the statement is reached, before deciding whether to repeat.

Why each choice is right or wrong

A - Incorrect: A for loop tests its condition before its first body execution and may execute zero times.

B - Correct: A do-while executes its body before testing the condition, guaranteeing the first body execution.

C - Incorrect: A while loop pre-tests; an initially false condition prevents all body executions.

D - Incorrect: For and while can execute zero times, so the first-execution guarantee does not apply to all forms.

Primary note proof

Study-note section OOP25

while and for pre-test and may execute zero times; do-while post-tests and executes once.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 7 | lines 26-41

Iteration: The for(), while() and
do...while() statements
4. Jumping Statements: to jump out
of loops and to control other areas of
program java uses break and
continue statements.
The break statement: used to exit a
loop
The Continue Statement: used to
force program to go back to the top
of a loop
Labels: Java does not include a goto
statement.
Instead of goto, Java allows you to
combine break and continue with a
label.

Question sources

Object-Oriented Programming practice questions · Question 51

Question 76 | Q00483

Which statement causes the loop to immediately jump to the next iteration of the loop?

A. Exit

B. Break

C. Jump

D. Continue

Hint

Trace condition tests, control transfers and printing as separate steps.

Answer D: Continue

continue skips the rest of the current iteration and proceeds to the loop's continuation step; break exits the loop. Thus continue matches moving to the next iteration.

Why each choice is right or wrong

A - Incorrect: Exit is not the named loop-jump statement here

B - Incorrect: break leaves the loop

C - Incorrect: Jump is a category, rather than the required keyword

D - Correct: continue advances control to loop continuation

Primary note proof

Study-note section OOP25

> break exits the loop; continue skips its remaining body and then runs a for update.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 7 | lines 26-41

> Iteration: The for(), while() and
> do...while() statements
> 4. Jumping Statements: to jump out
> of loops and to control other areas of
> program java uses break and
> continue statements.
> The break statement: used to exit a
> loop
> The Continue Statement: used to
> force program to go back to the top
> of a loop
> Labels: Java does not include a goto
> statement.
> Instead of goto, Java allows you to
> combine break and continue with a
> label.

Question sources

Object-Oriented Programming practice questions · Question 53

Question 77 | Q00484

A BREAK statement inside a Loop like WHILE, FOR, DO WHILE and Enhanced-FOR causes the program execution ___ Loop

A. Exit

B. Continuation with next iteration

C. Never exit

D. Can not say

Hint

Trace condition tests, control transfers and printing as separate steps.

Answer A: Exit

The break statement is explicitly defined as used to exit a loop.

Why each choice is right or wrong

A - Correct: Matches the stated effect

B - Incorrect: Describes continue rather than break

C - Incorrect: Contradicts the stated effect

D - Incorrect: The effect is directly stated

Primary note proof

Study-note section OOP25

> break exits the loop; continue skips its remaining body and then runs a for update.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 7 | lines 26-41

> Iteration: The for(), while() and
> do...while() statements
> 4. Jumping Statements: to jump out
> of loops and to control other areas of
> program java uses break and
> continue statements.
> The break statement: used to exit a
> loop
> The Continue Statement: used to
> force program to go back to the top
> of a loop
> Labels: Java does not include a goto
> statement.
> Instead of goto, Java allows you to
> combine break and continue with a
> label.

Question sources

Object-Oriented Programming practice questions · Question 54

Question 78 | Q00485

Java array is a collection of _____.

A. similar type of elements

B. different type of element

C. heterogeneous data

D. Both A and C

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer A: Elements with a common declared component type.

OOP26 says array elements share a declared common component type, possibly a superclass or interface. That is the precise interpretation of similar type in A. Reference elements may point to different compatible runtime subclasses, but this does not give the array unrelated declared element types or make arbitrary heterogeneous data its defining property.

Why each choice is right or wrong

A - Correct: Similar type is accurate when understood as the array's single declared common component type.

B - Incorrect: The array does not assign an unrelated declared type to each element; compatible runtime subclasses are a separate issue.

C - Incorrect: Heterogeneous runtime objects may share one component type, but unrestricted heterogeneity is not the array's defining type rule.

D - Incorrect: Combining A with unrestricted heterogeneity obscures the single declared component type required by OOP26.

Primary note proof

Study-note section OOP26

> Elements have a declared common component type, which may be a superclass/interface.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean

and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 55

Question 79 | Q00486

Array data access using _____.

A. Operator

B. Variable

C. index

D. Pointer

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer C: An index.

OOP26 defines valid array indices as zero through length-1. Selecting an array element therefore uses its index, as in a[i]. An operator participates in the syntax and a variable can hold the index, but neither term identifies the selection value as precisely. Java does not require a raw pointer to access an element.

Why each choice is right or wrong

A - Incorrect: Array-access syntax uses brackets, but operator is less specific than the index that selects the element.

B - Incorrect: A variable can store an index, but array selection also accepts other integer index expressions.

C - Correct: An index specifies the position of the requested element within the array’s valid bounds.

D - Incorrect: Java element access uses checked indices rather than requiring a programmer-visible raw pointer.

Primary note proof

Study-note section OOP26

Valid indices are 0..a.length-1; an invalid access throws ArrayIndexOutOfBoundsException.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 56

Question 80 | Q00487

Java Array can allocate _____.

A. Dynamic Memory

B. Static Memory

C. Both A and B

D. None of the above

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer A: Dynamic allocation of an array object.

OOP26 shows array creation through new int[5] and describes the result as an array object with fixed length after creation. Allocation occurs when that creation expression executes, which is the dynamic-memory meaning of A. Fixed array length does not mean statically allocated storage, and a static reference does not change the array object into static memory.

Why each choice is right or wrong

A - Correct: The creation expression allocates an array object when executed, even though the created object’s length is fixed.

B - Incorrect: A fixed length does not imply static allocation; new creates an array object during execution.

C - Incorrect: Declaring a static reference does not supply a second static-allocation mechanism for the array object itself.

D - Incorrect: The valid new-array expression in OOP26 demonstrates dynamic allocation, so none is false.

Primary note proof

Study-note section OOP26

int[] a=new int[5]; and int a[]={1,2,3}; are valid. int[] a={}; creates a zero-length array; explicit new is unnecessary in a declaration initializer. int a[],b; makes only a an array; int[] c,d; makes both arrays. Array objects have fixed length after creation, although the reference may be reassigned.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java
(byte, short, int, long, float, double,
char & Boolean)
2. Reference types: Arrays, classes,
and interfaces, String)
• Note: Boolean type has a value true or
false (there is no conversion b/n Boolean
and other types).
• All non-primitive types are reference
types, so all class types are reference
types.
• All integers in Java are signed, Java
doesn‟t support unsigned integers.

Object Oriented Programming module | PDF page 8 | lines 43-55

• a class declaration only creates a
template; it does not create an actual
object.
• To actually create a Dog object, use
the second statement.
• Dog myDog; // Creates only
the template
• myDog = new Dog(); //
create a Dog object called
myDog
• After this statement executes,

myDog will be an instance of
Dog.

Question sources

Object-Oriented Programming practice questions · Question 57

Question 81 | Q00488

Which complete Java array declaration has an incorrect initializer syntax? Treat each option independently.

A. int[] arr = new int[5];

B. int arr[] = new int[5];

C. int arr[] = new int[5];

D. int arr[] = int[5] new;

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer D: int arr[] = int[5] new;

Java permits array brackets on the declared type or after the variable name, and new int[5] allocates a five-element array. Thus A is valid, and B and its duplicate C are also valid declarations. D reverses the required allocation token order, putting new after int[5]. Replacing editorial periods with semicolons isolates the intended initializer-syntax error without treating punctuation noise as the tested concept.

Why each choice is right or wrong

A - Incorrect: Brackets on the type and new int[5] are valid.

B - Incorrect: Brackets after the variable name are legal with this allocation expression.

C - Incorrect: This duplicates B's valid declaration and is not an incorrect initializer.

D - Correct: The initializer must place new before int[5], so this token order is incorrect.

Primary note proof

Study-note section OOP26

int[] a=new int[5]; and int a[]={1,2,3}; are valid. int[] a={}; creates a zero-length array; explicit new is unnecessary in a declaration initializer. int a[],b; makes only a an array; int[] c,d; makes both arrays.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 58

Question 82 | Q00489

Index in array start with _____.

A. -1

B. 0

C. 1

D. infinite

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer B: 0.

OOP26 states that valid array indices run from zero to length minus one. Thus the first element of any nonempty Java array has index 0. An empty array has no valid element index, but that does not change the language’s zero-based indexing convention requested by this question.

Why each choice is right or wrong

A - Incorrect: Negative one is below the valid lower bound and causes an invalid access when used.

B - Correct: Zero is the first valid index of a nonempty Java array.

C - Incorrect: One selects the second element when present; Java arrays do not use one-based indexing.

D - Incorrect: Infinity is not an array index; valid positions are finite integers below the fixed array length.

Primary note proof

Study-note section OOP26

> Valid indices are 0..a.length-1; an invalid access throws ArrayIndexOutOfBoundsException.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 28 | lines 65-79

> ✓ Unchecked exceptions: Is an exception that occurs at the time of execution, these are also called as Runtime Exceptions.
> ✓ These include programming bugs, such as logic errors or improper use of an API.
> ✓ Runtime exceptions are ignored at the time of compilation.
> ✓ For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an ArrayIndexOutOfBoundsExceptione xception occurs.

Question sources

Object-Oriented Programming practice questions · Question 59

Question 83 | Q00490

Which shown Java declarations create and initialize a one-dimensional int array? Zero-length arrays count. Treat the options independently. Select all that apply.

A. int arr [] [] = {1, 2, 3, 4};

B. int [] arr = (1, 2, 3);

C. int [] arr = {};

D. int arr [] = {1, 2, 3};

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer C, D: int [] arr = {};; int arr [] = {1, 2, 3};

C creates an int array of length zero, and D creates one of length three initialized to 1, 2 and 3. Both declare, create and initialize a one-dimensional array. A declares two dimensions but supplies scalar elements instead of row arrays. B uses parentheses, which do not form an array initializer. Explicitly allowing zero length prevents incorrectly rejecting the valid empty array.

Why each choice is right or wrong

A - Incorrect: The flat int values do not initialize the declared int[][] component rows.

B - Incorrect: Parentheses do not provide a Java array initializer.

C - Correct: Empty braces validly create and initialize a zero-length int array.

D - Correct: The brace list validly creates and initializes a three-element int array.

Primary note proof

Study-note section OOP26

> int[] a=new int[5]; and int a[]={1,2,3}; are valid. int[] a={}; creates a zero-length array; explicit new is unnecessary in a declaration initializer. int a[],b; makes only a an array; int[] c,d; makes both arrays.

Study-note section OOP26

> Java two-dimensional arrays are arrays of arrays and may be ragged: {{12,-9,8},{7,14},{-32,-1,0}} has row lengths 3,2,3.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean
> and other types).
> • All non-primitive types are reference
> types, so all class types are reference
> types.
> • All integers in Java are signed, Java
> doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 60

Question 84 | Q00491

We can calculate the length of an array using _____.

A. sizeof(array)

B. array.len

C. array.length

D. array.sizeof()

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer C: array.length.

OOP26 uses a.length as the fixed element count when defining valid indices. Therefore array.length is the correct Java expression for the length field of an array reference named array. This is a field access rather than a method invocation, and the other supplied spellings are not the array-length mechanism described in the note.

Why each choice is right or wrong

A - Incorrect: sizeof is not the Java array-length expression used in OOP26's index-bound rule.

B - Incorrect: The field name is length, not len, so this spelling does not retrieve array length.

C - Correct: Java arrays expose their fixed element count through the length field, without parentheses.

D - Incorrect: The note uses a length field; arrays do not expose the proposed sizeof method.

Primary note proof

Study-note section OOP26

> Array objects have fixed length after creation, although the reference may be reassigned.

Study-note section OOP26

> Valid indices are 0..a.length-1; an invalid access throws ArrayIndexOutOfBoundsException.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean

and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 61

Question 85 | Q00492

Which is an inherent property of Java arrays? Here “Code Optimization” means a guarantee that merely introducing an array makes every algorithm run faster.

A. Code Optimization

B. Random access

C. Size No-Limit

D. Both A and B

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer B: Random access is the concrete array advantage.

Arrays provide indexed random access, so B states the inherent capability. They have a fixed length after creation and are bounded by implementation limits and available resources, so unlimited size is false. An array is a storage structure, not a guarantee that every algorithm becomes faster: allocating an unused array cannot improve a computation that does not need it. A therefore fails the stated guarantee, and combining it with B does not make D correct.

Why each choice is right or wrong

A - Incorrect: The stated universal speedup does not follow from array storage; adding an unused allocation does not improve the algorithm’s required computation.

B - Correct: Indexed random access is a direct array capability.

C - Incorrect: Array length is fixed after creation and constrained by implementation limits and resources.

D - Incorrect: This conjunction includes the false universal optimization guarantee, even though random access itself is true.

Primary note proof

Study-note section OOP27

Random access is the reliable array advantage; size is bounded by implementation/resources, not unlimited.

Study-note section OOP26

Array objects have fixed length after creation, although the reference may be reassigned.

Study-note section OOP03

Student s=new Student(); allocates and initializes an instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java
(byte, short, int, long, float, double,
char & Boolean)
2. Reference types: Arrays, classes,
and interfaces, String)
• Note: Boolean type has a value true or
false (there is no conversion b/n Boolean
and other types).
• All non-primitive types are reference
types, so all class types are reference
types.
• All integers in Java are signed, Java
doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 62

Question 86 | Q00493

Which term describes the logical index order of a Java array, from 0 through length−1, rather than making any claim about physical RAM placement?

A. Random

B. Sequential

C. Sequential & Random

D. Binary search

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer B: Sequential

Java arrays expose a logical sequence of indices starting at zero and ending at length minus one. Sequential describes that order. Random access describes the ability to choose an index directly, not a random logical ordering of elements or a portable physical-memory arrangement. Binary search is an algorithm rather than an ordering description. The revised question therefore tests index semantics without assuming exposed physical contiguity.

Why each choice is right or wrong

A - Incorrect: Random access does not mean that the logical index sequence is randomly ordered.

B - Correct: Sequential describes the logical index progression from 0 to length−1.

C - Incorrect: Combining access capability with ordering does not give the precise ordering term requested.

D - Incorrect: Binary search is an algorithm, not the array's logical index order.

Primary note proof

Study-note section OOP26

> Valid indices are 0..a.length-1; an invalid access throws ArrayIndexOutOfBoundsException.

Study-note section OOP27

> Random access is the reliable array advantage; size is bounded by implementation/resources, not unlimited.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean
> and other types).
> • All non-primitive types are reference
> types, so all class types are reference
> types.
> • All integers in Java are signed, Java
> doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 63

Question 87 | Q00494

What will be the output of the program?

```
class Main
{
public static void main(String args[]) {
int arr[] = {10, 20, 30, 40, 50};
for(int i=0; i < arr.length; i++)
{
System.out.print(" " + arr[i]);
}
}
}
```

A. 10 20 30 40 50

B. Compiler Error

C. 10 20 30 40

D. None of the above

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer D: Exact output: " 10 20 30 40 50", with an initial space.

The loop visits all five indices, 0 through 4. Each print prepends the literal string containing one space, including before the first element. The exact output is therefore " 10 20 30 40 50" without a newline. A lists the correct numbers but omits that initial space. Under literal output matching, D is the offered exact answer.

Why each choice is right or wrong

A - Incorrect: The listed values are right, but the printed option omits the leading space emitted before 10.

B - Incorrect: The array declaration and traversal stay within indices 0 through 4, so no compilation error follows.

C - Incorrect: The last index 4 is visited and prints 50; this also omits the initial space.

D - Correct: None of the other options reproduces the exact leading-space output required by the literal code.

Primary note proof

Study-note section OOP26

> int[] a=new int[5]; and int a[]={1,2,3}; are valid.

Study-note section OOP26

> Valid indices are 0..a.length-1; an invalid access throws ArrayIndexOutOfBoundsException.

Study-note section OOP25

while and for pre-test and may execute zero times; do-while post-tests and executes once.

Study-note section OOP24

Once a String operand is encountered, + concatenates in left-to-right groups: 1+2+"x" gives "3x", while "x"+1+2 gives "x12".

Study-note section OOP-S02

PrintStream.print(double) converts the stored value through String.valueOf and Double.toString. For finite nonzero magnitudes from 0.001 inclusive to 10000000 exclusive, the standard decimal form uses at least one fractional digit and only the further digits needed to distinguish the value from adjacent doubles. println(double) also adds a line terminator. This is not fixed-decimal-place formatting.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java
(byte, short, int, long, float, double,
char & Boolean)
2. Reference types: Arrays, classes,
and interfaces, String)
• Note: Boolean type has a value true or
false (there is no conversion b/n Boolean
and other types).
• All non-primitive types are reference
types, so all class types are reference
types.
• All integers in Java are signed, Java
doesn‟t support unsigned integers.

Object Oriented Programming module | PDF page 7 | lines 26-41

Iteration: The for(), while() and
do...while() statements
4. Jumping Statements: to jump out
of loops and to control other areas of
program java uses break and
continue statements.
The break statement: used to exit a
loop
The Continue Statement: used to
force program to go back to the top
of a loop
Labels: Java does not include a goto

statement.
Instead of goto, Java allows you to combine break and continue with a label.

Question sources

Object-Oriented Programming practice questions · Question 64

Question 88 | Q00495

Which of these is an incorrect Statement?

A. It is necessary to use new operator to initialize an array

B. Array can be initialized using comma separated expressions surrounded by curly braces

C. Array can be initialized when they are declared

D. None of the mentioned

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer A: Explicit new is not necessary in a declaration initializer.

OOP26 explicitly states that new need not be written in a declaration initializer and shows both empty and nonempty brace forms. Thus A is the incorrect statement. B and C describe the valid brace-initializer form used at declaration. This does not imply that an existing array variable can later be assigned a bare brace list.

Why each choice is right or wrong

A - Correct: The claim is false: brace initialization at declaration creates an array without explicitly writing new.

B - Incorrect: This is a true description of the comma-separated brace initializer shown in OOP26.

C - Incorrect: This is true: the note's examples initialize arrays as part of their declarations.

D - Incorrect: A is an offered false statement, so it is incorrect to say none is incorrect.

Primary note proof

Study-note section OOP26

int[] a=new int[5]; and int a[]={1,2,3}; are valid. int[] a={}; creates a zero-length array; explicit new is unnecessary in a declaration initializer.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java
(byte, short, int, long, float, double,
char & Boolean)
2. Reference types: Arrays, classes,
and interfaces, String)
• Note: Boolean type has a value true or
false (there is no conversion b/n Boolean
and other types).
• All non-primitive types are reference
types, so all class types are reference
types.
• All integers in Java are signed, Java
doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 65

Question 89 | Q00496

What is the type of variable "b" and "d" in the below snippet?
int a[], b;
int []c, d;

A. "b" and "d" are int

B. "b" and "d" are arrays of type int

C. "d" is int variable; and "b" is int array

D. "b" is int variable; and "d" is int array

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer D: b is int; d is int[].

In int a[],b, the brackets belong only to a, leaving b as a scalar int. In int[] c,d, the brackets are part of the common type, so both c and d are int arrays. OOP26 gives these exact forms. The distinction depends on bracket placement rather than on the variable names.

Why each choice is right or wrong

A - Incorrect: b is scalar int, but d belongs to the common int[] type of the second declaration.

B - Incorrect: Only d is an array; the first declaration attaches brackets to a rather than to b.

C - Incorrect: This reverses the actual types: b is scalar int and d is int[].

D - Correct: Brackets after a leave b scalar, while brackets after int make both c and d arrays.

Primary note proof

Study-note section OOP26

int a[],b; makes only a an array; int[] c,d; makes both arrays.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java
(byte, short, int, long, float, double,
char & Boolean)
2. Reference types: Arrays, classes,
and interfaces, String)
• Note: Boolean type has a value true or
false (there is no conversion b/n Boolean
and other types).
• All non-primitive types are reference
types, so all class types are reference
types.
• All integers in Java are signed, Java
doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 66

Question 90 | Q00497

How to sort an array?

A. Array.sort()

B. Arrays.sort()

C. Collection.sort()

D. System.sort()

Hint

Trace every element or API argument and distinguish copying values from aliasing a reference.

Answer B: Arrays.sort(array).

OOP27 specifies java.util.Arrays.sort(a) for in-place sorting of an array. B names that method, with the argument omitted as shorthand in the options; a real invocation supplies the array. The other class/method combinations do not match the array-sorting API provided here.

Why each choice is right or wrong

A - Incorrect: The supplied class name Array does not match the java.util.Arrays class specified here.

B - Correct: Arrays.sort is the correct method name; actual use supplies the array argument as Arrays.sort(a).

C - Incorrect: Collection.sort is not the array-sorting API identified in OOP27.

D - Incorrect: System.sort is not the array-sorting API provided in the complete note.

Primary note proof

Study-note section OOP27

> java.util.Arrays.sort(a) sorts in place.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean
> and other types).
> • All non-primitive types are reference
> types, so all class types are reference
> types.
> • All integers in Java are signed, Java
> doesn‟t support unsigned integers.

Object Oriented Programming module | PDF page 2 | lines 42-45

> • The API provides libraries, which
> groups a number of related classes
> and interfaces together into what is
> called a Java package.

Question sources

Object-Oriented Programming practice questions · Question 67

Question 91 | Q00498

Which correctly spelled Java API copies a selected range from one existing array into another existing array? The parentheses in distractors stand for a method name, not a complete no-argument invocation.

A. System.arraycopy(src, srcPos, dst, dstPos, length)

B. Array.copy()

C. Arrays.copy()

D. Collection.copy()

Hint

Trace every element or API argument and distinguish copying values from aliasing a reference.

Answer A: System.arraycopy(src, srcPos, dst, dstPos, length)

System.arraycopy copies length elements from the supplied source position into a supplied destination array position. Its name uses lowercase c in copy. Arrays.copyOf is a different supported API that creates a copy, but Arrays.copy is not that method's name. The other printed names do not identify the requested array-copy API. Correcting A's spelling and supplying its arguments restores a usable exact choice.

Why each choice is right or wrong

A - Correct: This is the correctly spelled range-copy API for existing source and destination arrays.

B - Incorrect: Array.copy is not the supported API named here.

C - Incorrect: The supported creation API is Arrays.copyOf, not Arrays.copy.

D - Incorrect: Collection.copy is not the specified array-range-copy API.

Primary note proof

Study-note section OOP27

> System.arraycopy(src,srcPos,dst,dstPos,length) copies into an existing array; its exact spelling uses lowercase c. Arrays.copyOf(a,n) creates a copy. a=b merely aliases the same array.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean

and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Object-Oriented Programming practice questions · Question 68

Question 92 | Q00499

Which named type categories can be directly declared as top-level members of a Java package? Subpackages are separately named packages, not automatically enclosed members. Select all that apply.

A. Classes

B. Sub Packages

C. Interfaces

D. All of the above

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer A, C: Classes; Interfaces

A package organizes named types, including classes and interfaces, and provides a naming and access boundary. A subpackage is a separate package identity even when its name starts with the parent’s name. It does not automatically gain membership or access through that spelling. Consequently classes and interfaces are the direct type categories, while the all option wrongly includes subpackages in the same membership sense.

Why each choice is right or wrong

A - Correct: Classes can be top-level named types belonging to the package.

B - Incorrect: A subpackage is a separate named package rather than a directly declared type member.

C - Correct: Interfaces can be top-level named types belonging to the package.

D - Incorrect: All includes the separate-package category and is therefore too broad.

Primary note proof

Study-note section OOP28

package p; precedes imports; imports precede top-level type declarations. import p.*; imports accessible types from p as needed, not types in p.sub automatically. Packages organize names

and control access; hierarchical spelling does not grant special access between a package and its subpackages.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 42-45

• The API provides libraries, which groups a number of related classes and interfaces together into what is called a Java package.

Question sources

Object-Oriented Programming practice questions · Question 69

Question 93 | Q00500

Which of these keywords is used to define packages in Java?

A. pkg

B. Pkg

C. package

D. Package

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer C: package

The source-file rules refer to a package statement as the first statement when a class belongs to a package; the provided spelling is package.

Why each choice is right or wrong

A - Incorrect: pkg is not the package-declaration keyword

B - Incorrect: Pkg is not the stated spelling

C - Correct: Matches the package statement spelling

D - Incorrect: Capitalized Package differs from the stated spelling

Primary note proof

Study-note section OOP28

package p; precedes imports; imports precede top-level type declarations.

Study-note section OOP28

Java identifiers are case sensitive; $ is allowed but @ is not an identifier character.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 51-70

2.3. Source file declaration
rules
• These rules are essential when declaring classes, import statements and package statements in a source file.
1. There can be only one public class per source file.
2. A source file can have multiple nonpublic classes.
3. The public class name should be the name of the source file as well which should be appended by .java at the end.
For example: the class name is public class Employee{} then the source file should be as Employee.java.
• If the class is defined inside a package, then the package statement should be the first statement in the source file.

Question sources

Object-Oriented Programming practice questions · Question 70

Question 94 | Q00501

An _____ statement can be used to access the classes and interface of a different package from the current package.

A. instanceOf

B. import

C. extends

D. implement

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer B: import

Import statements follow any package declaration and precede top-level type declarations; they come first when there is no package declaration. They bring external class/interface names into the source context.

Why each choice is right or wrong

A - Incorrect: instanceof is an object-type test, not an import

B - Correct: Matches the source-file import statement

C - Incorrect: extends defines inheritance

D - Incorrect: implement is not the import statement

Primary note proof

Study-note section OOP28

> package p; precedes imports; imports precede top-level type declarations. import p.*; imports accessible types from p as needed, not types in p.sub automatically. Packages organize names and control access; hierarchical spelling does not grant special access between a package and its subpackages.

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Study-note section OOP21

> Use if(a instanceof Ferrari) before narrowing when the dynamic type is uncertain.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 51-70

> 2.3. Source file declaration
> rules
> • These rules are essential when declaring
> classes, import statements and package
> statements in a source file.
> 1. There can be only one public
> class per source file.
> 2. A source file can have multiple
> nonpublic classes.
> 3. The public class name should be
> the name of the source file as
> well which should be appended
> by .java at the end.
> For example: the class name is public class
> Employee{} then the source file should be
> as Employee.java.
> • If the class is defined inside a
> package, then the package statement
> should be the first statement in the
> source file.

Object Oriented Programming module | PDF page 10 | lines 2-12

package statement and the class declaration.
• If there are no package statements then the import statement should be the first line in the source file.
• Import and package statements will imply to all the classes present in the source file.
• It is not possible to declare different import and/or package statements to different classes in the source file.

Question sources

Object-Oriented Programming practice questions · Question 71

Question 95 | Q00502

Which of the following packages is used to includes classes to create user interface like Button and Checkbox?

A. java.lang

B. java.net

C. java.awt

D. java.io

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer C: java.awt.

OOP28 explicitly identifies java.awt as including Button and Checkbox. The question names those UI classes, so the answer follows directly from their package membership. java.lang, networking and I/O packages serve other purposes and are not the package the note identifies for these two AWT controls.

Why each choice is right or wrong

A - Incorrect: java.lang is implicitly imported, but OOP28 locates Button and Checkbox in java.awt instead.

B - Incorrect: The named controls are AWT UI types, not networking types in the proposed java.net package.

C - Correct: OOP28 explicitly lists Button and Checkbox as classes included in java.awt.

D - Incorrect: java.io supplies input/output facilities rather than the two AWT controls named by the question.

Primary note proof

Study-note section OOP28

java.lang is implicitly imported; java.util supplies utilities; java.io supplies I/O; java.awt includes Button and Checkbox.

Study-note section OOP-S04

Calendar and Date are classes in java.util; Collection is an interface in that package. java.util also contains Arrays and Scanner. java.net supplies networking facilities; java.awt supplies the Button and Checkbox UI controls described in the package overview.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 27 | lines 20-41

• An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
• An exception is an indication of a problem that occurs during a program‟s execution.
• An exception is a runtime error.
• A Java exception is an instance of a class derived from Throwable.
• The Throwable class is contained in the java.lang package, and subclasses of Throwable are contained in various packages.
• Errors related to GUI components are included in the java.awt package; numeric exceptions are included in the java.lang package because they are related to the java.lang.Number class.
• You can create your own exception classes by extending Throwable or a subclass of Throwable.

Question sources

Object-Oriented Programming practice questions · Question 72

Question 96 | Q00503

Which of the following packages is used to includes utility classes like Calendar, Collection, Date?

A. java.lang

B. java.net

C. java.awt

D. java.util

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer D: java.util.

The question identifies Calendar, Collection and Date as utility types. OOP28 assigns utility facilities to java.util, in contrast to implicitly imported language facilities, AWT user-interface classes and I/O. Therefore java.util is the package category supported for the named utility examples, rather than the language, networking or UI alternatives.

Why each choice is right or wrong

A - Incorrect: java.lang supplies core language facilities; the utility package identified here is java.util.

B - Incorrect: java.net is the networking alternative, not the utility package identified in OOP28.

C - Incorrect: java.awt contains AWT UI controls, rather than the utility facilities requested by the question.

D - Correct: java.util is the note's explicit utility-package mapping for the class/interface examples in the stem.

Primary note proof

Study-note section OOP-S04

> Calendar and Date are classes in java.util; Collection is an interface in that package. java.util also contains Arrays and Scanner. java.net supplies networking facilities; java.awt supplies the Button and Checkbox UI controls described in the package overview.

Study-note section OOP28

> java.lang is implicitly imported; java.util supplies utilities; java.io supplies I/O; java.awt includes Button and Checkbox.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 42-45

> • The API provides libraries, which
> groups a number of related classes

and interfaces together into what is
called a Java package.

Question sources

Object-Oriented Programming practice questions · Question 73

Question 97 | Q00504

An accessible class declares an ordinary field. Which offered modifier choices permit another unrelated class in the same package to access that field directly? Choose the most complete answer; this asks about member modifiers, not top-level class modifiers.

A. Public

B. Protected

C. No Modifier

D. All of the mentioned

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer D: All of the mentioned

For ordinary members, public allows access, protected includes same-package access, and no modifier grants package access. All three listed choices therefore permit the unrelated same-package client to access the field. This does not claim that protected is legal on a top-level class, and class accessibility alone would not expose a private field. Under the explicitly member-level scope, D is the complete selection.

Why each choice is right or wrong

A - Incorrect: Public permits the access, but this is only one of the three valid modifier choices.

B - Incorrect: Protected includes same-package access, but this option alone is incomplete.

C - Incorrect: No modifier grants package access, but this option alone is incomplete.

D - Correct: All three listed member-modifier choices permit the specified same-package access.

Primary note proof

Study-note section OOP08

For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Study-note section OOP08

protected therefore does not mean “subclasses only”: unrelated classes in the same package also have access.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

Access Specifiers
Access level modifiers determine whether other classes can use a particular field or invoke a particular method.
Their are four types access levels:
1. No Modifier(Default): Visible to the package(source file), and it is called package-private
2. Private: Visible to the class only.
3. Public: Visible to the world.
4. Protected: Visible to the package and all subclasses.

Object Oriented Programming module | PDF page 9 | lines 51-70

2.3. Source file declaration rules
• These rules are essential when declaring classes, import statements and package statements in a source file.
1. There can be only one public class per source file.
2. A source file can have multiple nonpublic classes.
3. The public class name should be the name of the source file as well which should be appended by .java at the end.
For example: the class name is public class Employee{} then the source file should be as Employee.java.
• If the class is defined inside a package, then the package statement should be the first statement in the source file.

Question sources

Object-Oriented Programming practice questions · Question 74

Question 98 | Q00505

Which option is a syntactically complete Java on-demand import of accessible top-level types from package pkg, without importing subpackages automatically?

A. import pkg.

B. Import pkg.

C. import pkg.*;

D. Import pkg.*

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer C: import pkg.*;

A type-import-on-demand statement uses lowercase import, the package name, a dot and wildcard, and a semicolon. import pkg.*; therefore supplies the complete syntax. It makes accessible types from pkg available as needed without automatically importing types from pkg.sub. A lacks the wildcard or type name, and capitalized Import is not the Java keyword in B or D.

Why each choice is right or wrong

A - Incorrect: The trailing dot lacks a wildcard or type name and a complete terminator.

B - Incorrect: Capitalized Import is not the keyword, and the target is incomplete.

C - Correct: This is the complete lowercase wildcard import statement.

D - Incorrect: Capitalized Import is not a Java keyword and the statement lacks its semicolon.

Primary note proof

Study-note section OOP28

> import p.*; imports accessible types from p as needed, not types in p.sub automatically.

Study-note section OOP28

> Java identifiers are case sensitive; $ is allowed but @ is not an identifier character.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 10 | lines 2-12

> package statement and the class
> declaration.
> • If there are no package statements
> then the import statement should be
> the first line in the source file.
> • Import and package statements will
> imply to all the classes present in the

source file.
• It is not possible to declare different import and/or package statements to different classes in the source file.

Object Oriented Programming module | PDF page 27 | lines 20-41

• An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
• An exception is an indication of a problem that occurs during a program‟s execution.
• An exception is a runtime error.
• A Java exception is an instance of a class derived from Throwable.
• The Throwable class is contained in the java.lang package, and subclasses of Throwable are contained in various packages.
• Errors related to GUI components are included in the java.awt package; numeric exceptions are included in the java.lang package because they are related to the java.lang.Number class.
• You can create your own exception classes by extending Throwable or a subclass of Throwable.

Question sources

Object-Oriented Programming practice questions · Question 75

Question 99 | Q00506

Which of the following is false statement about package in java?

A. Packages are used for preventing naming conflicts

B. Providing controlled access: protected and default have package level access control.

C. Packages cannot be considered as data encapsulation

D. Both B and C

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer C: The claim that packages cannot support encapsulation is false.

OOP28 says packages organize names and control access, while OOP08 defines package access and same-package protected access. These mechanisms can hide implementation types or members behind permitted boundaries, so C's categorical denial is false. A and B describe real package roles. D incorrectly includes the true package-access statement B among the false claims.

Why each choice is right or wrong

A - Incorrect: Organizing names into packages supports avoiding naming conflicts, so this is a true purpose rather than the false claim.

B - Incorrect: Default access is package access, and protected also grants same-package access; the statement is true.

C - Correct: Packages control access and can hide implementation types or members, contradicting this categorical denial of encapsulation.

D - Incorrect: B is supported by the access rules, so both B and C cannot be classified as false.

Primary note proof

Study-note section OOP28

> Packages organize names and control access; hierarchical spelling does not grant special access between a package and its subpackages.

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Study-note section OOP08

> protected therefore does not mean "subclasses only": unrelated classes in the same package also have access.

Study-note section OOP02

> Information hiding keeps implementation decisions behind a stable interface.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 42-45

> • The API provides libraries, which
> groups a number of related classes
> and interfaces together into what is
> called a Java package.

Object Oriented Programming module | PDF page 13 | lines 62-73

> Access Specifiers
> Access level modifiers determine whether

other classes can use a particular field or
invoke a particular method.
Their are four types access levels:
1. No Modifier(Default): Visible to
the package(source file), and it is
called package-private
2. Private: Visible to the class only.
3. Public: Visible to the world.
4. Protected: Visible to the package
and all subclasses.

Question sources

Object-Oriented Programming practice questions · Question 76

Question 100 | Q00507

For a Java naming hierarchy such as p and p.q, what is the specific conventional term for packages such as p.q relative to p?

A. packages

B. nested packages sharing their parent's member-access scope

C. util subpackages

D. subpackages

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer D: Subpackages.

The conventional term for p.q relative to p is subpackage. This is a naming hierarchy, not a lexical enclosing scope: the parent's import does not automatically import types from the subpackage, and their related names grant no special member access. Packages is only the broad category. The util qualifier would restrict the answer to a particular package family instead of naming the general relationship.

Why each choice is right or wrong

A - Incorrect: Packages names the broad category but does not identify the requested relationship to a parent name.

B - Incorrect: The naming hierarchy does not give a subpackage its parent's member-access scope.

C - Incorrect: The relationship applies to arbitrary package names, not just util subpackages.

D - Correct: Subpackages is the conventional term for the requested naming relationship.

Primary note proof

Study-note section OOP28

import p.*; imports accessible types from p as needed, not types in p.sub automatically. Packages organize names and control access; hierarchical spelling does not grant special access between a package and its subpackages.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 42-45

• The API provides libraries, which groups a number of related classes and interfaces together into what is called a Java package.

Question sources

Object-Oriented Programming practice questions · Question 77

Question 101 | Q00508

_____ can be defined as the process where one class acquires the properties (methods and fields) of another.

A. Overriding

B. Inheritance

C. Polymorphism

D. Abstraction

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer B: Inheritance

Inheritance means acquiring another class's methods and fields.

Why each choice is right or wrong

A - Incorrect: Overrides a method rather than acquiring the class properties generally

B - Correct: Matches the definition word for word

C - Incorrect: Many forms is a different principle

D - Incorrect: Exposes essential information rather than acquiring class properties

Primary note proof

Study-note section OOP13

class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Study-note section OOP13

Use inheritance when a subtype can satisfy the parent's contract: Dog is an Animal.

Study-note section OOP02

Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Study-note section OOP14

A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype.

Study-note section OOP02

Abstraction presents the essential operations while omitting details irrelevant to the caller.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

What is Inheritance?
• Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{

```
.....}
class Sub extends Super{ }
```

Question sources

Object-Oriented Programming practice questions · Question 78

Question 102 | Q00509

The class which inherits the properties of other is known as _____

A. superclass

B. parent class

C. subclass

D. None of the above

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer C: subclass

The class that does the inheriting is explicitly called subclass, derived class or child class.

Why each choice is right or wrong

A - Incorrect: Superclass is the source class

B - Incorrect: Parent class is the source class

C - Correct: Subclass is the inheriting class

D - Incorrect: C identifies the inheriting class correctly

Primary note proof

Study-note section OOP13

class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

What is Inheritance?
• Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one

class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Question sources

Object-Oriented Programming practice questions · Question 79

Question 103 | Q00510

Subclass also known as ?

A. derived class

B. child class

C. base class

D. Both A and B

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer D: Both A and B

Subclass, derived class and child class are synonyms; base class is a superclass synonym.

Why each choice is right or wrong

A - Incorrect: A valid subclass synonym

B - Incorrect: A valid subclass synonym

C - Incorrect: Names the superclass instead

D - Correct: Combines both valid synonyms

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

> What is Inheritance?
> • Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
> • Inheritance is the process where one class acquires the properties (methods and fields) of another.
> • A class that is inherited is called a superclass.
> • The class that does the inheriting is called a subclass.
> • OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
> • The class whose properties are inherited is known as superclass (base class, parent class).
> • Therefore, a subclass is a specialized version of a superclass.
> • It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
> • The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
> • Sytax:
> class Super{
> .....}
> class Sub extends Super{ }

Question sources

Object-Oriented Programming practice questions · Question 80

Question 104 | Q00511

_____ is the keyword used to inherit the properties of a class.

A. inherit

B. poly

C. extends

D. super

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer C: extends

The inheritance syntax is class Sub extends Super.

Why each choice is right or wrong

A - Incorrect: Not the stated keyword

B - Incorrect: Not the stated keyword

C - Correct: Explicitly identified inheritance keyword

D - Incorrect: Refers to the superclass rather than declaring inheritance

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Study-note section OOP16

> super.name selects a hidden superclass field; super.sound() directly requests the parent implementation.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

> What is Inheritance?
> • Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
> • Inheritance is the process where one

class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Question sources

Object-Oriented Programming practice questions · Question 81

Question 105 | Q00512

The super keyword is similar to _____ keyword.

A. construct

B. this

C. class

D. extends

Hint

Distinguish the current instance, parent context, hidden fields and static dispatch.

Answer B: this

super accesses superclass members, while this refers to the current object; the comparison concerns member access, rather than distinct receiver objects.

Why each choice is right or wrong

A - Incorrect: construct is not the compared keyword

B - Correct: Explicitly named comparison

C - Incorrect: class defines a type rather than this-like access

D - Incorrect: extends establishes inheritance rather than member reference

Primary note proof

Study-note section OOP16

> Java this denotes the current object; this.name=name distinguishes a field from a parameter. super.name selects a hidden superclass field; super.sound() directly requests the parent implementation. It does not create a second parent object. An unqualified instance call may dispatch dynamically; an explicit super call is different.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 17 | lines 51-74

> The super keyword
> • Since encapsulation is a primary attribute of OOP, Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.
> • Super has two general forms.
> 1. Used to access parent class instance members.
> 2. Used to access parent class constructor.
> 1. Used to access parent class instance members.
> • super acts somewhat like this, except that it always refers to the superclass of the subclass in which it is used.
> • Super has the form: super.member Here, member can be either a method or an instance variable.
> • Super is most applicable to resolve member hiding.
> • super( ) must always be the first statement executed inside a subclass‟ constructor.

Question sources

Object-Oriented Programming practice questions · Question 82

Question 106 | Q00513

Which of these is correct way of inheriting class A by class B?

A. class B + class A {}

B. class B inherits class A {}

C. class B extends A {}

D. class B extends class A {}

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer C: class B extends A {}

The declared subclass comes first, followed by extends and the superclass name; the example is class Sub extends Super.

Why each choice is right or wrong

A - Incorrect: Plus is not the inheritance syntax

B - Incorrect: inherits is not the inheritance keyword

C - Correct: Correct order and single class keyword

D - Incorrect: Extra class before the parent violates the demonstrated form

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

> What is Inheritance?
> • Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
> • Inheritance is the process where one class acquires the properties (methods and fields) of another.
> • A class that is inherited is called a superclass.
> • The class that does the inheriting is called a subclass.

• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Question sources

Object-Oriented Programming practice questions · Question 83

Question 107 | Q00514

A class which is declared with the _____ keyword is known as an abstract class in Java.

A. abstract

B. util

C. extends

D. None of the above

Hint

Check whether a class is concrete and whether every inherited abstract contract has an implementation.

Answer A: abstract

The abstract-class definition explicitly uses the abstract keyword.

Why each choice is right or wrong

A - Correct: Matches the definition

B - Incorrect: util is not the modifier

C - Incorrect: extends names an inheritance relation

D - Incorrect: A directly supplies the keyword

Primary note proof

Study-note section OOP18

> An abstract class cannot be directly instantiated but can define fields, constructors, abstract methods and concrete methods. It need not contain an abstract method. An abstract method has no body; abstract and final conflict because an operation cannot require overriding while forbidding it.

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 23 | lines 57-81

> Abstract Classes
> • A classes declared with abstract
> keyword and can‟t be instantiated.
> • Used to hide meaningless methods:
> • There can be no objects of an abstract
> class. .:.
> • An abstract class cannot be directly
> instantiated with the new operator.
> • You cannot declare abstract
> constructors, or abstract static methods
> • Syntax: abstract class class_name
> Example:
> abstract class A {
> abstract void callme();
> // concrete methods are still
> allowed in abstract classes
> void callmetoo() {
> System.out.println("This is a
> concrete method.");
> }}
> class B extends A {
> void callme() {
> System.out.println("B's
> implementation of callme.");
> }}

Question sources

Object-Oriented Programming practice questions · Question 84

Question 108 | Q00515

A method which is declared as abstract and does not have implementation is known as an _____?

A. Abstract Interface

B. Abstract Thread

C. Abstract List

D. abstract Method

Hint

Check whether a class is concrete and whether every inherited abstract contract has an implementation.

Answer D: abstract Method

The summary says abstract methods have no body/implementation. That is the method kind described by the stem.

Why each choice is right or wrong

A - Incorrect: Confuses a method with an interface

B - Incorrect: Thread is not the unimplemented method category

C - Incorrect: List is not the unimplemented method category

D - Correct: Matches a method without implementation

Primary note proof

Study-note section OOP18

> An abstract method has no body; abstract and final conflict because an operation cannot require overriding while forbidding it.

Study-note section OOP19

> interface Payable{double getPaymentAmount();} declares an abstract public operation. class Invoice implements Payable must provide public double getPaymentAmount(), perhaps returning getQuantity()*getPricePerItem().

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 24 | lines 63-73

> Summery:
> Abstract:
> • Classes: declared with abstract
> keyword and can‟t be instantiated.
> • Methods: without body (no
> implementation)

Final:
• Class: declared as final prevents inheritance.
• Method: declared as final prevents overriding that method.

Question sources

Object-Oriented Programming practice questions · Question 85

Question 109 | Q00516

Which offered access modifier may be explicitly applied to an ordinary top-level Java interface? This is not a member interface nested in another type.

A. Public

B. private

C. Protected

D. All of the mentioned

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer A: Public

An ordinary top-level interface may be public or package-private. Of the explicit access modifiers listed, only public is permitted at top level. Private and protected can apply in some nested member contexts but are excluded by the stated declaration location. No modifier is the package-private alternative, though it is not one of the offered choices. Therefore the aggregate of all modifiers is false.

Why each choice is right or wrong

A - Correct: Public is permitted on an ordinary top-level interface.

B - Incorrect: Private is not a legal top-level interface modifier.

C - Incorrect: Protected is not a legal top-level interface modifier.

D - Incorrect: All includes the forbidden private and protected top-level modifiers.

Primary note proof

Study-note section OOP08

Top-level classes/interfaces may be public or package-private, not protected/private. Member types have additional visibility possibilities.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 33 | lines 31-46

The d/c b/n Interface & Class
• You cannot instantiate an interface
• Interface has not constructor
• Methods in an interface are all abstract.
• Fields in an interface are all static and final
• An interface can extend multiple interfaces
Summery
• By default, methods in interface are public and abstract.
• Data members are public, static and final.
• An interface with empty body is called Tag/marker interface.

Question sources

Object-Oriented Programming practice questions · Question 86

Question 110 | Q00517

Which of these keywords is used by a class to use an interface defined previously?

A. Import

B. import

C. implements

D. Implements

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer C: implements

A class implements an interface, while an interface extends another interface.

Why each choice is right or wrong

A - Incorrect: Import is neither the demonstrated spelling nor implementation keyword

B - Incorrect: import is a source naming facility, not implementation

C - Correct: Matches class implementation syntax

D - Incorrect: Capitalized form is not the stated keyword

Primary note proof

Study-note section OOP19

interface Payable{double getPaymentAmount();} declares an abstract public operation. class Invoice implements Payable must provide public double getPaymentAmount(), perhaps returning getQuantity()*getPricePerItem(). A class can extend a superclass and implement zero or more interfaces. An interface can extend several interfaces.

Study-note section OOP28

package p; precedes imports; imports precede top-level type declarations. import p.*; imports accessible types from p as needed, not types in p.sub automatically. Packages organize names and control access; hierarchical spelling does not grant special access between a package and its subpackages.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 31 | lines 9-30

• It provides total abstraction;
• Means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default.
• A class that implement interface must implement all the methods declared in the interface.
• If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
Rules of Interface
• Variables declared in an interface are public, static & final by default.
• Java does not allow you to extend more than one class, However you can implement more than one interfaces in your class.
• class implements interface but an interface extends another interfaces.

Question sources

Object-Oriented Programming practice questions · Question 87

Question 111 | Q00518

Which modifier lists can legally prefix int X = 1; when it is declared as a field in a Java interface? Missing public/static/final modifiers are supplied implicitly. Treat each choice independently. Select all that apply.

A. public static

B. private final

C. public final

D. static final

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer A, C, D: public static; public final; static final

Every interface field is implicitly public static final. A can omit final, C can omit static, and D can omit public because the language supplies each missing modifier. The initializer int X=1 satisfies the constant field's initialization requirement. Private conflicts with the field's public visibility and is therefore forbidden. The three legal partial modifier lists all produce the same effective field modifiers.

Why each choice is right or wrong

A - Correct: public static is legal because final is implicit.

B - Incorrect: private conflicts with an interface field's implicit public visibility.

C - Correct: public final is legal because static is implicit.

D - Correct: static final is legal because public is implicit.

Primary note proof

Study-note section OOP19

> No interface instance fields or constructors exist; fields are implicitly public static final.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 31 | lines 9-30

> • It provides total abstraction;
> • Means all the methods in interface
> are declared with empty body and
> are public and all fields are public,
> static and final by default.
> • A class that implement interface
> must implement all the methods
> declared in the interface.
> • If a class implements an interface

and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
Rules of Interface
• Variables declared in an interface are public, static & final by default.
• Java does not allow you to extend more than one class, However you can implement more than one interfaces in your class.
• class implements interface but an interface extends another interfaces.

Question sources

Object-Oriented Programming practice questions · Question 88

Question 112 | Q00519

In Java 17, which option most completely describes the permitted method contents of an interface? Here “declaration” means an abstract method declaration without a body and “definition” means a permitted method with a body; it does not ask what every interface must contain.

A. Method definition

B. Method declaration

C. Method declaration and definition

D. Method name

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer C: Method declaration and definition

Modern Java interfaces can contain abstract method declarations and method bodies of permitted kinds, including default and static methods and private helpers. They need not contain every kind, but both declarations and definitions are allowed. A and B each describe only part of that capability. C is the most complete category, while a bare method name does not constitute the required declaration or definition.

Why each choice is right or wrong

A - Incorrect: Permitted method bodies are part of the answer, but this omits abstract declarations.

B - Incorrect: Abstract declarations are allowed, but this omits permitted method bodies.

C - Correct: Both abstract declarations and permitted method definitions can appear in Java 17 interfaces.

D - Incorrect: A name alone is not a complete method declaration or definition.

Primary note proof

Study-note section OOP19

A method declaration without private, default or static is implicitly public abstract. Modern interfaces may also have default/static method bodies, and Java 9+ private helper methods.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 30 | lines 37-59

Chapter - Five: Java Interface
• An interface in java is a blueprint of a class.
• It has static constants and abstract methods.
• The interface in java is a mechanism to achieve abstraction.
• There can be only abstract methods in the java interface not method body (contain only signature, not body).
• It is used to achieve abstraction and multiple inheritance in Java.
• Java Interface also represents IS-A relationship.
• It cannot be instantiated just like abstract class.
• There are mainly three reasons to use interface.
• It is used to achieve abstraction.
• By interface, we can support the functionality of multiple inheritance.

Object Oriented Programming module | PDF page 33 | lines 31-46

The d/c b/n Interface & Class
• You cannot instantiate an interface
• Interface has not constructor
• Methods in an interface are all abstract.
• Fields in an interface are all static and final
• An interface can extend multiple

interfaces
Summery
• By default, methods in interface are public and abstract.
• Data members are public, static and final.
• An interface with empty body is called Tag/marker interface.

Question sources

Object-Oriented Programming practice questions · Question 89

Question 113 | Q00520

What type of methods an interface contain by default?

A. abstract

B. static

C. final

D. private

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer A: abstract

The summary explicitly says interface methods are public and abstract by default.

Why each choice is right or wrong

A - Correct: Matches the stated default

B - Incorrect: Not the default method category in this context

C - Incorrect: Not the stated default

D - Incorrect: Contradicts the public default

Primary note proof

Study-note section OOP19

A method declaration without private, default or static is implicitly public abstract.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 33 | lines 31-46

The d/c b/n Interface & Class
• You cannot instantiate an interface
• Interface has not constructor
• Methods in an interface are all abstract.
• Fields in an interface are all static and final
• An interface can extend multiple interfaces
Summery
• By default, methods in interface are public and abstract.
• Data members are public, static and final.
• An interface with empty body is called Tag/marker interface.

Question sources

Object-Oriented Programming practice questions · Question 90

Question 114 | Q00521

When does Exceptions in Java arises in code sequence?

A. Run Time

B. Can Occur Any Time

C. Compilation Time

D. None of the mentioned

Hint

Separate source errors from runtime exceptions and determine the hierarchy and checked obligation.

Answer A: Exceptions are thrown at runtime; checked obligations are checked at compile time.

OOP29 explains that checked does not mean an exception occurs during compilation. The compiler verifies catch-or-declare obligations; an actual exception throw occurs during program execution. Therefore Run Time is the correct occurrence answer. Compile-time syntax or name errors are a different category and do not make Java exceptions occur at any arbitrary stage.

Why each choice is right or wrong

A - Correct: An actual exception throw occurs during execution, as OOP29 explicitly states.

B - Incorrect: Any time conflates runtime throwing with compile-time checking and source errors.

C - Incorrect: Compilation checks obligations or reports source errors; it is not when the described program throws an exception.

D - Incorrect: Run Time is offered and is the supported occurrence stage, so none is false.

Primary note proof

Study-note section OOP29

A missing semicolon or undeclared variable is a compile-time source error, not a checked exception.

Study-note section OOP29

Checked does not mean the exception occurs at compile time: the compiler checks an obligation to catch or declare it, while an actual throw occurs at runtime.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 27 | lines 20-41

• An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
• An exception is an indication of a problem that occurs during a program‟s execution.
• An exception is a runtime error.
• A Java exception is an instance of a class derived from Throwable.
• The Throwable class is contained in the java.lang package, and subclasses of Throwable are contained in various packages.
• Errors related to GUI components are included in the java.awt package; numeric exceptions are included in the java.lang package because they are related to the java.lang.Number class.
• You can create your own exception classes by extending Throwable or a subclass of Throwable.

Object Oriented Programming module | PDF page 28 | lines 25-43

1. Checked exception: Is an exception that occurs at the compile time,
✓ These are also called as compile time exceptions.

✓ These exceptions cannot simply be ignored at the time of compilation,
✓ The Programmer should take care of (handle) these exceptions.
✓ For example, if you use FileReader class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a FileNotFoundException occurs, and compiler prompts the programmer to handle the exception.

Object Oriented Programming module | PDF page 28 | lines 65-79

✓ Unchecked exceptions: Is an exception that occurs at the time of execution, these are also called as Runtime Exceptions.
✓ These include programming bugs, such as logic errors or improper use of an API.
✓ Runtime exceptions are ignored at the time of compilation.
✓ For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an ArrayIndexOutOfBoundsExceptione xception occurs.

Question sources

Object-Oriented Programming practice questions · Question 91

Question 115 | Q00522

Which of these keywords is not a part of exception handling?

A. finally

B. thrown

C. catch

D. try

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer B: thrown

try, catch and finally are Java exception-handling keywords; thrown is an English word in compiler diagnostics, rather than a Java keyword.

Why each choice is right or wrong

A - Incorrect: Present in the example's cleanup block

B - Correct: Not a keyword used to declare or handle exceptions; distinguish the word thrown in diagnostics

C - Incorrect: Present as the handler clause

D - Incorrect: Present as the monitored block

Primary note proof

Study-note section OOP30

> throw new IOException("message") actively throws a value; throws IOException in a signature declares a possible checked exception. A caller must handle or propagate checked obligations. try encloses guarded operations; a matching catch handles the thrown type or a subtype. Execution skips the remainder of the try when a throw occurs.

Study-note section OOP31

> finally normally executes after try and any selected catch, including when leaving by return or propagating an exception.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 28 | lines 8-22

```
try {
int num=
reader.nextInt();
System.out.println("The number
is :" + num);
}
catch
(InputMismatchException e){
System.out.println("There is
type mis match :");
}
finally{
System.out.println("This part is
always done");
}} }
```

Object Oriented Programming module | PDF page 29 | lines 11-22

Using try and catch...
General form of try/catch:
Try{
//block of code to monitor for
errors
}
Catch(ExceptionType1 exOb) {
//handler for ExceptionType1
}
Catch(ExceptionType2 exOb) {
//handler for ExceptionType2
}

Object Oriented Programming module | PDF page 28 | lines 25-43

1. Checked exception: Is an
exception that occurs at the compile
time,
✓ These are also called as compile
time exceptions.
✓ These exceptions cannot simply
be ignored at the time of
compilation,
✓ The Programmer should take
care of (handle) these exceptions.
✓ For example, if you use
FileReader class in your program
to read data from a file, if the file
specified in its constructor
doesn't exist, then a
FileNotFoundException occurs,
and compiler prompts the
programmer to handle the
exception.

Question sources

Object-Oriented Programming practice questions · Question 92

Object-Oriented Programming practice questions · Question 93

Question 116 | Q00524

Which of these keywords must be used to handle the exception thrown by try block in some rational manner?

A. finally

B. throw

C. catch

D. try

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer C: catch

The catch clause contains the handler for its exception type. finally is separate cleanup, and try surrounds the monitored operation.

Why each choice is right or wrong

A - Incorrect: Runs final code rather than the type-specific handler

B - Incorrect: Would signal an exception rather than handle one

C - Correct: The explicit exception handler clause

D - Incorrect: Marks monitored code, not the handler

Primary note proof

Study-note section OOP30

> throw new IOException("message") actively throws a value; throws IOException in a signature declares a possible checked exception. A caller must handle or propagate checked obligations. try encloses guarded operations; a matching catch handles the thrown type or a subtype. Execution skips the remainder of the try when a throw occurs.

Study-note section OOP31

> finally normally executes after try and any selected catch, including when leaving by return or propagating an exception.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 28 | lines 8-22

```
try {
int num=
reader.nextInt();
System.out.println("The number
is :" + num);
}
```

```
catch
(InputMismatchException e){
System.out.println("There is
type mis match :");
}
finally{
System.out.println("This part is
always done");
}} }
```

Object Oriented Programming module | PDF page 29 | lines 11-22

```
Using try and catch...
General form of try/catch:
Try{
//block of code to monitor for
errors
}
Catch(ExceptionType1 exOb ) {
//handler for ExceptionType1
}
Catch(ExceptionType2 exOb ) {
//handler for ExceptionType2
}
```

Question sources

Object-Oriented Programming practice questions · Question 94

Object-Oriented Programming practice questions · Question 95

Question 117 | Q00526

Which of these is a super class of all errors and exceptions in the Java language?

A. Catchable

B. Throwable

C. RunTimeExceptions

D. None of the above

Hint

Separate source errors from runtime exceptions and determine the hierarchy and checked obligation.

Answer B: Throwable.

OOP29 places Exception and Error under the common root Throwable. Therefore Throwable is the superclass covering all errors and exceptions in the question. RuntimeException is only a sub-branch and cannot be the root of Error or checked exceptions. Catchable is not the root type named in the provided hierarchy.

Why each choice is right or wrong

A - Incorrect: Catchable is not the root class specified here’s throwable hierarchy.

B - Correct: Throwable is explicitly the shared root above both Exception and Error.

C - Incorrect: RuntimeException is an unchecked exception branch, not the common superclass of all errors and exceptions.

D - Incorrect: Throwable is offered and is the hierarchy root, so none of the above is false.

Primary note proof

Study-note section OOP29

> Throwable is the root of throwable values, with Exception and Error branches. RuntimeException and Error subclasses are unchecked; other throwable classes are checked.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 27 | lines 20-41

> • An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
> • An exception is an indication of a problem that occurs during a program‟s execution.
> • An exception is a runtime error.
> • A Java exception is an instance of a class derived from Throwable.
> • The Throwable class is contained in the java.lang package, and subclasses of Throwable are contained in various packages.
> • Errors related to GUI components are included in the java.awt package; numeric exceptions are included in the java.lang package because they are related to the java.lang.Number class.
> • You can create your own exception classes by extending Throwable or a subclass of Throwable.

Object Oriented Programming module | PDF page 29 | lines 2-10

Exception Vs Errors
1. Errors: An Error indicates serious problem that a reasonable application should not try to catch.
2. Exception: Exception indicates conditions that a reasonable application might try to catch.

Question sources

Object-Oriented Programming practice questions · Question 96

Question 118 | Q00527

In which of the following package Exception class exist?

A. java.file

B. java.lang

C. java.io

D. java.util

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer B: java.lang.Exception belongs to java.lang.

The base Exception class belongs to java.lang, which is implicitly imported. Its place in the Throwable hierarchy does not determine the package of every subclass: I/O-specific subclasses may belong to java.io while the base remains java.lang.Exception. java.util contains utility types, and java.file is not the package of this class. Thus B supplies the correct package.

Why each choice is right or wrong

A - Incorrect: The official API places Exception in java.lang, not the proposed java.file package.

B - Correct: The official Java SE17 API explicitly identifies Exception as a java.lang class.

C - Incorrect: Specific I/O exceptions do not relocate the Exception base class from java.lang into java.io.

D - Incorrect: java.util provides utility types, while the authoritative API places Exception in java.lang.

Primary note proof

Study-note section OOP-S04

The base Exception class is java.lang.Exception. It does not belong to java.io merely because some subclasses represent I/O failures. Package membership and position in an inheritance hierarchy are separate facts.

Study-note section OOP29

Throwable is the root of throwable values, with Exception and Error branches.

Study-note section OOP28

java.lang is implicitly imported; java.util supplies utilities; java.io supplies I/O; java.awt includes Button and Checkbox.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 27 | lines 20-41

• An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
• An exception is an indication of a problem that occurs during a program‟s execution.
• An exception is a runtime error.
• A Java exception is an instance of a class derived from Throwable.
• The Throwable class is contained in the java.lang package, and subclasses of Throwable are contained in various packages.
• Errors related to GUI components are included in the java.awt package; numeric exceptions are included in the java.lang package because they are related to the java.lang.Number class.
• You can create your own exception classes by extending Throwable or a subclass of Throwable.

Question sources

Object-Oriented Programming practice questions · Question 97

Question 119 | Q00528

Applets are designed to be embedded within an _____.

A. Javascript

B. Css

C. HTML

D. SQL

Hint

Use the historical host lifecycle and distinguish activation, suspension, drawing and unloading.

Answer C: An HTML document, in the historical applet model.

OOP34 explicitly says historical applet code was embedded in an HTML document and executed by a compatible JVM host/plugin or applet viewer. HTML is therefore the requested embedding format. The answer concerns legacy applets; it does not imply modern browsers execute applets or that JavaScript, CSS or SQL provides the described host document.

Why each choice is right or wrong

A - Incorrect: JavaScript is not the embedding document format identified for historical applets in OOP34.

B - Incorrect: CSS specifies styling rather than the HTML document into which the historical applet is embedded.

C - Correct: HTML is the document format explicitly named as the historical applet embedding context.

D - Incorrect: SQL is not the document format that hosts the historical applet in OOP34.

Primary note proof

Study-note section OOP34

> In the historical applet model, Java code was embedded in an HTML document and executed by a compatible JVM host/plugin or applet viewer.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 8 | lines 32-36

> • Java classes do not need to have a
> main( ) method except the starting
> point for your program.
> • Further, applets don‟t require a main(
>) method at all.

Question sources

Object-Oriented Programming practice questions · Question 98

Question 120 | Q00529

Which of the following is required to view an applet?

A. JCM

B. JDM

C. JVM

D. Java class

Hint

Use the historical host lifecycle and distinguish activation, suspension, drawing and unloading.

Answer C: A JVM, together with a compatible historical host or applet viewer.

OOP34 says applets execute through a compatible JVM host/plugin or applet viewer. JVM is therefore the necessary component among the named options, although a JVM alone is not the entire browser embedding environment. This is a historical requirement rather than a statement that current browsers provide applet support.

Why each choice is right or wrong

A - Incorrect: JCM is not the JVM execution component specified by OOP34 for historical applet viewing.

B - Incorrect: JDM is not the JVM host or applet-viewer component identified here.

C - Correct: A compatible JVM is necessary to execute the applet, together with an appropriate historical host/viewer.

D - Incorrect: A Java class supplies code but does not itself provide the required JVM execution environment.

Primary note proof

Study-note section OOP34

> In the historical applet model, Java code was embedded in an HTML document and executed by a compatible JVM host/plugin or applet viewer.

Study-note section OOP34

> A JVM alone is also not the complete browser embedding environment.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 8 | lines 32-36

> • Java classes do not need to have a
> main( ) method except the starting
> point for your program.

• Further, applets don‟t require a main(
) method at all.

Object Oriented Programming module | PDF page 2 | lines 17-29

The Java compiler translates the program into an intermediate language called Java bytecodes.
Java bytecodes are platform-independent.
Guaranteed to be Write Once, Run Anywhere.
There are different Java interpreters for different platforms.
Java is: + Object Oriented + Platform Independent
+ Simple + Secure
+ Portable +
Multithreaded + Robust

Question sources

Object-Oriented Programming practice questions · Question 99

Question 121 | Q00530

Which method is automatically called after the browser calls the init method?

A. start

B. stop

C. destroy

D. paint

Hint

Use the historical host lifecycle and distinguish activation, suspension, drawing and unloading.

Answer A: start.

OOP34 says init initializes once and start follows initialization and resumes activity. Thus start is the lifecycle method called after init under the historical host model. stop suspends activity, destroy cleans up before unloading, and paint draws content; none states the next lifecycle stage after initialization.

Why each choice is right or wrong

A - Correct: start follows initialization and resumes applet activity, exactly matching the requested lifecycle point.

B - Incorrect: stop suspends activity when deactivated; it is not the post-init startup callback.

C - Incorrect: destroy performs cleanup before unloading, rather than starting activity immediately after initialization.

D - Incorrect: paint draws content and is not the lifecycle startup callback that follows init.

Primary note proof

Study-note section OOP34

> init initializes once; start follows initialization and resumes activity; stop suspends activity when the host deactivates it; destroy performs final cleanup before unloading. paint draws content and is not a constructor.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 8 | lines 32-36

> • Java classes do not need to have a
> main( ) method except the starting
> point for your program.
> • Further, applets don‟t require a main(
>) method at all.

Question sources

Object-Oriented Programming practice questions · Question 100

Question 122 | Q00531

In the historical Java applet lifecycle, which callback performs final cleanup when a compatible host normally unloads the applet?

A. start

B. stop

C. destroy

D. paint

Hint

Use the historical host lifecycle and distinguish activation, suspension, drawing and unloading.

Answer C: destroy

The applet lifecycle assigns final cleanup before unloading to destroy. start begins or resumes activity, stop handles host deactivation, and paint draws content. A normal browser shutdown may cause an unload, but unloading can also occur without shutting the browser down. The revised stem therefore asks for the cleanup callback directly and does not make an incorrect exclusive shutdown claim or promise cleanup after an abrupt crash.

Why each choice is right or wrong

A - Incorrect: start begins or resumes activity rather than performing final cleanup.

B - Incorrect: stop suspends activity on host deactivation, not final unloading.

C - Correct: destroy performs final cleanup before normal applet unloading.

D - Incorrect: paint draws content rather than handling final cleanup.

Primary note proof

Study-note section OOP34

> init initializes once; start follows initialization and resumes activity; stop suspends activity when the host deactivates it; destroy performs final cleanup before unloading. paint draws content and is not a constructor.

Study-note section OOP34

> destroy is not defined exclusively as “browser shuts down”: the host may destroy an applet on unloading, and abrupt process termination does not guarantee cleanup.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 8 | lines 32-36

> • Java classes do not need to have a
> main( ) method except the starting
> point for your program.
> • Further, applets don‟t require a main(
>) method at all.

Question sources

Object-Oriented Programming practice questions · Question 101

Question 123 | Q00566

What are non-static Java fields declared directly in a class, outside methods, constructors and initializer-block local scopes, called?

A. Instance variables

B. Local variables

C. Class variables

D. Static variables

Hint

Locate the declaration scope and check default initialization versus definite assignment.

Answer A: Instance variables

Fields belong to a class declaration, but non-static fields hold separate state for each object and are called instance variables. Static fields are class variables shared by instances, so merely being outside a method would not distinguish them. The stem now explicitly excludes static and block-local declarations. Local variables belong to method, constructor or block scopes rather than to each object's field state.

Why each choice is right or wrong

A - Correct: Non-static object fields are instance variables.

B - Incorrect: Locals belong to executable scopes rather than the class's object state.

C - Incorrect: Class variables are static fields, excluded by the question.

D - Incorrect: Static variables are shared class fields, excluded by the question.

Primary note proof

Study-note section OOP05

> The introductory three-way grouping is local, instance and class/static variables. Locals occur in methods, constructors or blocks; they must be definitely assigned before use and cannot have public/private/protected modifiers. Fields occur in a class outside method bodies; non-static fields belong to each instance, while static fields belong to the class. Declaration location alone does not distinguish instance and static fields.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 5 | lines 14-38

> B. Instance variables:
> • Are declared in a class, but
> outside a method, constructor or
> any block.
> • Have default values.
> • are variables which are bound to
> the object itself. Meaning=>
> • Every instance of that class
> (object) has it's own copy of that
> variable
> • Are created when an object is
> created with the use of the
> keyword 'new' and destroyed
> when the object is destroyed.
> • Can be declared in class level
> before or after use.
> • Access modifiers can be given
> for instance variables.
> • Are visible for all methods,

constructors and block in the class.
E.g. class MyClass {
float aFloat; }
NB: By default, a member declared within a class is an instance member.

Object Oriented Programming module | PDF page 5 | lines 39-61

C. Class/Static variables
• Are declared with the static keyword in a class, but outside a method, constructor or a block.
• There would only be one copy of each class variable per class, regardless of how many objects are created from it.
• Are rarely used other than being declared as constants. (public static final)
• Are stored in the static memory.
• Static variables are created when the program starts and destroyed when the program stops.
• Visibility is similar to instance variables. However, most static variables are declared public
• Default values are same as instance variables.
• Static variables can be accessed by calling with the class name ClassName.VariableName.

Question sources

Computer Science model exam · 2015 · Question 35

Question 124 | Q00571

Which one of the following sending a message to an object is correct?

A. obj.add

B. obj.add.obj

C. obj.add()

D. obj.add().obj

Hint

Separate parameters from arguments and operations that read state from operations that change it.

Answer C: obj.add()

An instance method is invoked with object.method(arguments), for example ob.test(). Sending the add message with no arguments follows obj.add().

Why each choice is right or wrong

A - Incorrect: Member selection without invocation parentheses

B - Incorrect: Does not form the demonstrated call syntax

C - Correct: Uses object receiver, method name and argument parentheses

D - Incorrect: Further .obj is not part of the demonstrated message invocation

Primary note proof

Study-note section OOP11

> An instance method can be invoked as object.method(), or with an implicit this inside suitable instance context.

Study-note section OOP01

> Objects send messages through method calls such as obj.add().

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 11 | lines 24-41

```
class OverloadDemo {
void test() {
System.out.println("No
parameters"); }
void test(int a) { // Overload
test for one integer parameter.
System.out.println("a: " + a); }
void test(int a, int b) { //
Overload test for two integer
parameters.
System.out.println("a and b: " +
a + " " + b); }
double test(double a) { //
overload test for a double
parameter
System.out.println("double a: "
+ a);
return a*a; } }
```

Object Oriented Programming module | PDF page 10 | lines 62-72

Methods
• Are members of classes and do operations.
• Allows different classes to communicate each other.
• E.g. type name(parameter-list)
{ // body of method
//....
return value; // if type is not void

Question sources

Computer Science model exam · 2015 · Question 40

Question 125 | Q00576

The process of binding both attributes and methods together within a class is?

A. Inheritance

B. Encapsulation

C. Polymorphism

D. Overriding

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer B: Encapsulation

Binding attributes and methods into one class is encapsulation.

Why each choice is right or wrong

A - Incorrect: Transfers properties across classes

B - Correct: Groups data and the operations on that data

C - Incorrect: Allows an operation to take different forms

D - Incorrect: Replaces an inherited method implementation

Primary note proof

Study-note section OOP02

Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface.

Study-note section OOP02

Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects. Prefer a precise mechanism over a vague slogan: wrapping fields and methods points to encapsulation; showing what an operation does while hiding how points to abstraction.

Study-note section OOP14

A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 35-45

ፎ. Encapsulation:
• means the wrapping up of data and methods into a single unit (a class).
• It also means that the data inside a class is hidden from everything outside the class.
• The data can only be accessed by invoking the methods of the class.
• It is placing the data and the functions that work on that data in the same place.

Question sources

Computer Science model exam · 2015 · Question 45

Question 126 | Q00587

A keyword used to have only a single piece of storage for a particular field is?

A. this

B. object

C. Final

D. Static

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer D: Static

Only one copy of each class variable exists per class regardless of object count, and static is the declaration modifier for that property.

Why each choice is right or wrong

A - Incorrect: this does not provide shared storage

B - Incorrect: object does not declare the shared field category

C - Incorrect: final prevents modification rather than determining one copy

D - Correct: Static denotes the intended static modifier, though the literal Java spelling is lowercase static

Primary note proof

Study-note section OOP07

> static gives one field per loaded class, shared by its instances. If count starts at zero and every constructor increments it, three creations leave count=3. Prefer ClassName.count and ClassName.utility().

Study-note section OOP07

> final prevents reassignment of a variable, overriding of a method or extension of a class. A final reference may still refer to a mutable object.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 5 | lines 39-61

> C. Class/Static variables
> • Are declared with the static keyword in a class, but outside a method, constructor or a block.
> • There would only be one copy of each class variable per class, regardless of how many objects are created from it.
> • Are rarely used other than being declared as constants. (public static final)
> • Are stored in the static memory.
> • Static variables are created when the program starts and destroyed when the program stops.
> • Visibility is similar to instance variables. However, most static variables are declared public
> • Default values are same as instance variables.
> • Static variables can be accessed by calling with the class name ClassName.VariableName.

Question sources

Computer Science model exam · 2015 · Question 56

Question 127 | Q00589

What represents the concept that data can only be manipulated in known, well-defined ways

A. Inheritance

B. Information hiding

C. Polymorphism

D. overloading

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer B: Information hiding

Encapsulation hides class data and controls access through the class's methods. This is the information-hiding control over how data is manipulated.

Why each choice is right or wrong

A - Incorrect: Does not restrict manipulation paths

B - Correct: Describes control through the hidden-data/method boundary

C - Incorrect: Concerns multiple forms rather than restricting access

D - Incorrect: Concerns same-name methods with differing parameters

Primary note proof

Study-note section OOP02

> Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Study-note section OOP12

> Overloaded methods share a name but have different parameter types, counts or type order.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 35-45

ፎ. Encapsulation:
• means the wrapping up of data and methods into a single unit (a class).
• It also means that the data inside a class is hidden from everything outside the class.
• The data can only be accessed by invoking the methods of the class.
• It is placing the data and the functions that work on that data in the same place.

Question sources

Computer Science model exam · 2015 · Question 58

Question 128 | Q00599

Which one of the following is incorrect about inheritance

A. Subclass inherits members of superclass

B. Constructors are not members

C. Constructors of superclass can be invoked from subclass

D. Constructors of superclass inherited by subclass

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer D: Constructors of superclass inherited by subclass

Java constructors are not members and are not inherited, but a subclass can invoke its superclass constructor using super.

Why each choice is right or wrong

A - Incorrect: Subclass membership inheritance is the general mechanism

B - Incorrect: Constructors are not members.

C - Incorrect: Explicitly demonstrated with super(arg-list)

D - Correct: Contradicts the noninheritance rule

Primary note proof

Study-note section OOP-S05

In Java’s class-declaration terminology, constructors are not members and are not inherited. A subclass can invoke superclass construction; this does not turn the superclass constructor into an inherited subclass method.

Study-note section OOP10

Under Java 17 rules, an explicit this(args) or super(args) invocation must be the first constructor statement. this invokes another constructor in the same class; super invokes the immediate superclass constructor. A constructor cannot explicitly invoke both. If there is no explicit delegation, an implicit super() is inserted; a matching accessible no-argument parent constructor must exist.

Study-note section OOP13

class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 16 | lines 51-70

Types of inheritance
1. Simple/single inheritance: is damn easy to understand. When a class extends another one class only.
2. Multi-Level inheritance: One can inherit from a derived class, thereby making this derived class the base class for the new class.
3. Hierarchical inheritance: One class is inherited by many sub classes.
• Java does not support the inheritance of multiple super classes into a single subclass. (Multiple & Hybrid inheritances)
• No class can be a superclass of itself
• Superclass can be used by itself.
• The superclass has no knowledge of what a subclass adds to it.
• Constructors are not members, so they are not inherited by subclasses.

Object Oriented Programming module | PDF page 18 | lines 36-47

2. Used to access parent class constructor.
• A subclass can call a constructor defined by its superclass by use of the following form of super:
super(arg-list);
• Here, arg-list specifies any arguments needed by the constructor in the superclass.
• super( ) must always be the first

statement executed inside a subclass‟
constructor.

Question sources

Computer Science model exam · 2015 · Question 68

Question 129 | Q00605

Consider an abstract Java class that explicitly declares an abstract method. Which statement is incorrect about that class? In C, constructing objects means directly instantiating this abstract class.

A. It is a class containing abstract method

B. Abstract method is incomplete

C. Used to construct objects

D. None of the above

Hint

Check whether a class is concrete and whether every inherited abstract contract has an implementation.

Answer C: Used to construct objects

The given class contains an abstract method by assumption, and that method has no implementation body. An abstract class may include constructors and concrete methods, but it cannot itself be directly instantiated. C is therefore the false statement in the stated sense. A describes the given class rather than imposing a requirement on every abstract class, and None is false because C is incorrect.

Why each choice is right or wrong

A - Incorrect: This is true of the particular class because the stem explicitly says it declares an abstract method.

B - Incorrect: An abstract method has no implementation body, which is the intended meaning of incomplete.

C - Correct: Direct instantiation of the abstract class is prohibited, so this is the incorrect statement.

D - Incorrect: C is an incorrect statement, so None of the above cannot be selected.

Primary note proof

Study-note section OOP18

An abstract class cannot be directly instantiated but can define fields, constructors, abstract methods and concrete methods. It need not contain an abstract method. An abstract method has

no body; abstract and final conflict because an operation cannot require overriding while forbidding it.

Study-note section OOP18

A concrete subclass must provide or inherit implementations of all required abstract methods; otherwise it must itself be abstract.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 23 | lines 57-81

Abstract Classes
• A classes declared with abstract keyword and can‟t be instantiated.
• Used to hide meaningless methods:
• There can be no objects of an abstract class. .:.
• An abstract class cannot be directly instantiated with the new operator.
• You cannot declare abstract constructors, or abstract static methods
• Syntax: abstract class class_name
Example:
abstract class A {
abstract void callme();
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
}}
class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
}}

Object Oriented Programming module | PDF page 24 | lines 63-73

Summery:
Abstract:
• Classes: declared with abstract keyword and can‟t be instantiated.
• Methods: without body (no implementation)
Final:
• Class: declared as final prevents inheritance.

• Method: declared as final prevents overriding that method.

Question sources

Computer Science model exam · 2015 · Question 74

Question 130 | Q00608

Which statement correctly describes imports in an ordinary Java source compilation unit?

A. A Java source file may contain zero or more import statements before its top-level type declarations

B. Import statement appears after the class declaration

C. Import statement appears before package statement

D. All of the above

Hint

Check declaration order, exact case, package boundaries and the compile-versus-launch stage.

Answer A: A Java source file may contain zero or more import statements before its top-level type declarations

An optional package declaration comes first, followed by any import declarations, then the top-level type declarations. A source file can have no imports or multiple imports in that position. B places imports after a class, and C places them before the package declaration, reversing the ordering rules. D includes those false statements. These ordering rules apply to Java source files, rather than compiled .class contents.

Why each choice is right or wrong

A - Correct: This places zero or more source imports before the top-level types.

B - Incorrect: Imports precede top-level class declarations rather than following them.

C - Incorrect: The package declaration, when present, precedes imports.

D - Incorrect: B and C reverse the required ordering, so all is incorrect.

Primary note proof

Study-note section OOP28

package p; precedes imports; imports precede top-level type declarations.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 51-70

2.3. Source file declaration
rules
• These rules are essential when declaring classes, import statements and package statements in a source file.
1. There can be only one public class per source file.
2. A source file can have multiple nonpublic classes.
3. The public class name should be the name of the source file as well which should be appended by .java at the end.
For example: the class name is public class Employee{} then the source file should be as Employee.java.
• If the class is defined inside a package, then the package statement should be the first statement in the source file.

Object Oriented Programming module | PDF page 10 | lines 2-12

package statement and the class declaration.
• If there are no package statements then the import statement should be the first line in the source file.
• Import and package statements will imply to all the classes present in the source file.
• It is not possible to declare different import and/or package statements to different classes in the source file.

Question sources

Computer Science model exam · 2015 · Question 77

Question 131 | Q00618

Which one of the following is correct about narrowing casting?

A. Done by the programmer

B. It is automatic

C. Done by the compiler

D. All

Hint

Identify the primitive category, width and conversion rule before evaluating a declaration or expression.

Answer A: Done by the programmer

Narrowing is explicit and must be done programmatically using a cast; the automatic conversion section instead describes widening.

Why each choice is right or wrong

A - Correct: Matches explicit programmatic narrowing

B - Incorrect: Confuses narrowing with widening

C - Incorrect: The compiler does not insert the narrowing cast described here

D - Incorrect: B and C do not match explicit narrowing

Primary note proof

Study-note section OOP06

> A compatible widening conversion often needs no cast; narrowing usually needs an explicit cast and may lose information. Widening an int to float can lose precision, so “widening never loses information” is false.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 42-55

> 2. Narrowing (Explicit)
> Casting: if you want to
> narrow or go in the other
> direction, Java won‟t do that
> automatically. You have to
> do that programmatically.
> Syntax: smallerDT
> var=(smallerDT)largerDTVar
> OR largerValue;
> Example: float y=78.9f; byte
> b = (byte)y; //explicit type
> casting required
> int x = (int)2.7; double d =
> 100.04; long l = (long)d;

Question sources

Computer Science model exam · 2015 · Question 87

Question 132 | Q00626

Which one of the following is incorrect about method overriding?

A. A method declared final cannot be overridden

B. Similar argument list is required

C. Constructors cannot be overridden

D. None

Hint

Check inheritance, signature compatibility, visibility and eligibility for an override.

Answer D: None

final methods cannot be overridden, argument types and order must match, and constructors are not inherited methods. Hence none of A-C is incorrect under the intended exact-match reading.

Why each choice is right or wrong

A - Incorrect: Explicitly prohibited by overriding rules

B - Incorrect: Here, “similar argument list” must mean the same argument types in the same order.

C - Incorrect: Constructors are not inherited methods and have the class's own name

D - Correct: None of A-C contradicts these rules

Primary note proof

Study-note section OOP14

> A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype. Visibility cannot become more restrictive, and checked exceptions cannot be broadened incompatibly. @Override requests compiler checking; Java has no override keyword. Constructors, final methods, static methods and private parent methods are not overridden.

Study-note section OOP09

> Constructors are not inherited or overridden.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 21 | lines 26-39

> Rules of method overriding in Java..
> • Argument list: The argument list of
> overriding (method of subclass) and
> Overridden (method of super class)
> method must be similar.
> • The data types of the arguments and

their sequence should exactly match.
• Method names: must be similar.
• private, static and final methods cannot be overridden as they are local to the class.
• Access Modifier of the overriding method cannot be more restrictive than the overridden method of parent class.

Object Oriented Programming module | PDF page 16 | lines 51-70

Types of inheritance
1. Simple/single inheritance: is damn easy to understand. When a class extends another one class only.
2. Multi-Level inheritance: One can inherit from a derived class, thereby making this derived class the base class for the new class.
3. Hierarchical inheritance: One class is inherited by many sub classes.
• Java does not support the inheritance of multiple super classes into a single subclass. (Multiple & Hybrid inheritances)
• No class can be a superclass of itself
• Superclass can be used by itself.
• The superclass has no knowledge of what a subclass adds to it.
• Constructors are not members, so they are not inherited by subclasses.

Object Oriented Programming module | PDF page 12 | lines 24-47

Constructors
• Performs automatic initialization of objects. .:.
• Initializes an object immediately upon creation.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
• The default constructor automatically initializes all instance variables to zero and all object references to NULL.
• The default constructor is often

sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
• Once you define your own constructor, the default constructor is no longer used.
• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Question sources

Computer Science model exam · 2015 · Question 95

Question 133 | Q00641

Which listed cases are not checked exceptions in Java? Interpret “array index out of bounds” as ArrayIndexOutOfBoundsException and “file required not found” as FileNotFoundException. Select all that apply.

A. Missing semicolons

B. Array index out of bounds

C. Usage of undeclared variables

D. File required not found

Hint

Separate source errors from runtime exceptions and determine the hierarchy and checked obligation.

Answer A, B, C: Missing semicolons; Array index out of bounds; Usage of undeclared variables

A missing semicolon and an undeclared variable are source errors rejected during compilation; neither is a checked exception. ArrayIndexOutOfBoundsException is a runtime exception and is unchecked. FileNotFoundException is a checked exception, so its catch-or-declare obligation is checked by the compiler even though an actual throw occurs at runtime. The revised multiple-selection question accepts all three cases outside the checked-exception category.

Why each choice is right or wrong

A - Correct: A missing semicolon is a compile-time source error, not a checked exception.

B - Correct: ArrayIndexOutOfBoundsException is unchecked.

C - Correct: An undeclared variable is a compile-time source error, not a checked exception.

D - Incorrect: FileNotFoundException is checked and therefore is not selected.

Primary note proof

Study-note section OOP29

IOException, FileNotFoundException and SQLException are standard checked examples. ArithmeticException, NullPointerException, ClassCastException and array bounds exceptions are unchecked. A missing semicolon or undeclared variable is a compile-time source error, not a checked exception.

Study-note section OOP29

Checked does not mean the exception occurs at compile time: the compiler checks an obligation to catch or declare it, while an actual throw occurs at runtime.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 28 | lines 25-43

1. Checked exception: Is an exception that occurs at the compile time,
✓ These are also called as compile time exceptions.
✓ These exceptions cannot simply be ignored at the time of compilation,
✓ The Programmer should take care of (handle) these exceptions.
✓ For example, if you use FileReader class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a FileNotFoundException occurs, and compiler prompts the programmer to handle the exception.

Object Oriented Programming module | PDF page 28 | lines 65-79

✓ Unchecked exceptions: Is an exception that occurs at the time of execution, these are also called as Runtime Exceptions.
✓ These include programming bugs, such as logic errors or improper use of an API.
✓ Runtime exceptions are ignored at the time of compilation.
✓ For example, if you have declared an

array of size 5 in your program, and trying to call the 6th element of the array then an ArrayIndexOutOfBoundsExceptione xception occurs.

Question sources

Computer Science model exam · 2015 · Question 110

Question 134 | Q00670

If a class is named Student, the class constructor name is_____..

A. Any legal java identifier

B. Any legal java identifier starting with s

C. Student

D. Student constructor

Hint

Check the class name, return-type absence and the constructors actually declared.

Answer C: Student

The constructor must have the same name as its class, so a Student constructor is Student.

Why each choice is right or wrong

A - Incorrect: An arbitrary identifier need not match the class

B - Incorrect: Starting with s is not sufficient and need not match Student

C - Correct: Exactly matches the class name

D - Incorrect: Adds an extra word not in the class name

Primary note proof

Study-note section OOP09

A constructor has the class's exact name and no return type, not even void.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 12 | lines 24-47

Constructors
• Performs automatic initialization of objects. .:.
• Initializes an object immediately

upon creation.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
• The default constructor automatically initializes all instance variables to zero and all object references to NULL.
• The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
• Once you define your own constructor, the default constructor is no longer used.
• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Question sources

Computer Science model exam 2 · Question 11

Question 135 | Q00671

A visibility modifier that makes methods and data fields accessible only from within its own class.

A. Default

B. protected

C. public

D. private

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer D: private

The access table defines private as visible to the class only.

Why each choice is right or wrong

A - Incorrect: Default includes the package

B - Incorrect: Protected includes package and subclasses

C - Incorrect: Public is visible to the world

D - Correct: Matches class-only visibility

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

> Access Specifiers
> Access level modifiers determine whether
> other classes can use a particular field or
> invoke a particular method.
> Their are four types access levels:
> 1. No Modifier(Default): Visible to
> the package(source file), and it is
> called package-private
> 2. Private: Visible to the class only.
> 3. Public: Visible to the world.
> 4. Protected: Visible to the package
> and all subclasses.

Question sources

Computer Science model exam 2 · Question 12

Question 136 | Q00672

The keyword used to create subclass/derived class in inheritance is_____

A. Implement

B. Implements

C. extend

D. extends

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer D: extends

The syntax is class Sub extends Super, using the exact lowercase keyword extends.

Why each choice is right or wrong

A - Incorrect: Not the class inheritance keyword

B - Incorrect: Not the class inheritance keyword

C - Incorrect: Missing the final s

D - Correct: Matches the stated extends spelling

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

> What is Inheritance?
> • Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
> • Inheritance is the process where one class acquires the properties (methods and fields) of another.
> • A class that is inherited is called a superclass.
> • The class that does the inheriting is called a subclass.
> • OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
> • The class whose properties are inherited is known as superclass (base class, parent class).
> • Therefore, a subclass is a specialized version of a superclass.
> • It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
> • The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
> • Sytax:
> class Super{
> .....}
> class Sub extends Super{ }

Question sources

Computer Science model exam 2 · Question 13

Question 137 | Q00673

Which of the following is not valid variable in java?

A. A$_2

B. By2

C. Java

D. A@2$

Hint

Check the lexical category and whether the notation names a value, a declaration or non-executed source text.

Answer D: A@2$ is invalid.

OOP28 explicitly allows $ in Java identifiers but rejects @ as an identifier character. D contains @, so it cannot be the requested variable name. A's dollar sign and underscore do not create that defect. By2 and Java also contain ordinary letters and a permitted trailing digit where present.

Why each choice is right or wrong

A - Incorrect: A$_2 uses a letter first and permitted dollar sign, underscore and digit thereafter.

B - Incorrect: By2 starts with a letter and has an allowed digit after the letters.

C - Incorrect: Java is a case-sensitive ordinary identifier, not the prohibited @ pattern identified in the note.

D - Correct: A@2$ contains @, which OOP28 explicitly prohibits within a Java identifier.

Primary note proof

Study-note section OOP-S01

> For ordinary ASCII Java identifiers, the first character is a letter, underscore or dollar sign; later characters may also be digits. A reserved keyword is not an ordinary identifier, and a lone underscore is not a valid identifier in Java 17.

Study-note section OOP28

> Java identifiers are case sensitive; $ is allowed but @ is not an identifier character.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 11-20

> ✓ Reserved keywords: class,
> extends, implements, for...

✓ Identifiers: programmers given name; grade, gRade... ✓ Literals (Constants values): 55, 60.78, “Hi”, „M“... Java language specifies five major types of literals => integer, floating, character, string & Boolean literals

Question sources

Computer Science model exam 2 · Question 14

Question 138 | Q00674

A blue print of an object is called _____.

A. operation

B. method

C. class

D. Attribute

Hint

Distinguish an object’s stored state, operations, identity and class definition.

Answer C: class

A class is the blueprint or template for an object.

Why each choice is right or wrong

A - Incorrect: Operation describes behavior

B - Incorrect: Method implements an operation

C - Correct: The stated blueprint concept

D - Incorrect: Attribute describes state

Primary note proof

Study-note section OOP01

An object has identity, state and behavior. State is stored in fields; behavior is provided by methods. A Rectangle object may hold width and height and answer getArea() or getPerimeter(). Two objects can have equal fields and still be distinct objects. A class defines their common structure and behavior; an object is a particular instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 2-18

• Are the basic runtime entities in an OO program.
• Are identifiable things or individuals?
• Are of significance to the system
• Have states reflected in the attributes (data) held about the object
• Have characteristic behaviour (i.e. functionality)
ይ. Class:
• Is a blueprint for an object.
• is the definition of a set of objects
• which are all similar in terms of their attributes
• A class consists of a class name, data and methods.

Object Oriented Programming module | PDF page 8 | lines 4-20

2.1. Fundamentals of Classes
• A class is a blue print from which individual objects are created.
• Class is a template for an object, and an object is an instance of a class
• Class is the logical construct upon which the entire Java language is built because it defines the shape and nature of an object.
• As such, the class forms the basis for object-oriented programming in Java.
• Classes usually consist of two things: instance variables and methods
• Any concept you wish to implement in a Java program must be encapsulated within a class.

Question sources

Computer Science model exam 2 · Question 15

Question 139 | Q00743

How is generalization implemented in Object-Oriented software engineering?

A. Inheritance

B. Polymorphism

C. Encapsulation

D. Abstract Classes

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer A: Inheritance

Generalization groups specialized classes under a general superclass; a subclass specializes its superclass, and inheritance implements that hierarchy.

Why each choice is right or wrong

A - Correct: Creates the generalized/specialized class hierarchy

B - Incorrect: Uses multiple forms within a hierarchy rather than creating its relationship

C - Incorrect: Groups data and methods

D - Incorrect: Can serve as a base but is not the general implementation mechanism required

Primary note proof

Study-note section OOP13

> Use inheritance when a subtype can satisfy the parent's contract: Dog is an Animal.

Study-note section OOP-S05

> Aggregation models one instance grouping other instances as a whole with parts. It is an object-relationship concept, not a method-dispatch rule. Inheritance instead relates a specialized class to a general class; overridden operations can then provide polymorphic behavior.

Study-note section OOP02

> Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

> What is Inheritance?
> • Inheritance is one of the cornerstones of
> object-oriented programming because it

allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Object Oriented Programming module | PDF page 3 | lines 46-56

5. Inheritance:
• It is the way in which objects of one class get the properties of objects of another class(data &methods).
• For example, Person, employee and retiree
• Inheritance provides the idea of reusability - the Person class can be reused without making changes to it, because another class can be made to inherit from it.

Question sources

Computer Science model exam 2 · Question 84

Question 140 | Q00747

Choose the incorrect statement in terms of Objects.

A. Objects are abstractions of real-world

B. Objects can't manage themselves

C. Objects encapsulate state and representation information

D. All of the mentioned

Hint

Distinguish an object’s stored state, operations, identity and class definition.

Answer B: “Objects cannot manage themselves” is the incorrect generalization.

OOP01 says objects hold state and behavior, control responsibilities, communicate through method calls and can delegate. It expressly calls the claim that objects cannot manage their own state misleading. Therefore B is the false generalization. A is a conventional modeling description, and C describes encapsulated state, so all statements cannot be incorrect.

Why each choice is right or wrong

A - Incorrect: Objects can model abstractions of real-world entities; this conventional modeling description is not the identified false generalization.

B - Correct: Objects can control their state through behavior and delegate responsibilities, contradicting this categorical claim.

C - Incorrect: Encapsulation of state and related representation is consistent with the note’s object model.

D - Incorrect: The supported abstraction and encapsulation descriptions prevent treating every supplied statement as incorrect.

Primary note proof

Study-note section OOP01

> An object has identity, state and behavior. State is stored in fields; behavior is provided by methods. A Rectangle object may hold width and height and answer getArea() or getPerimeter(). Two objects can have equal fields and still be distinct objects. A class defines their common structure and behavior; an object is a particular instance.

Study-note section OOP01

> Choose classes around responsibilities: an account controls its own balance, a transaction records a payment, and a service coordinates them. Objects send messages through method calls such as obj.add(). They can delegate work to other objects; saying that objects cannot manage their own state is misleading. Procedural and object oriented designs both support reuse; global shared data is not compulsory in procedural programming.

Study-note section OOP02

Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 2-18

• Are the basic runtime entities in
an OO program.
• Are identifiable things or
individuals?
• Are of significance to the system
• Have states reflected in the
attributes (data) held about
the object
• Have characteristic behaviour
(i.e. functionality)
፪. Class:
• Is a blueprint for an object.
• is the definition of a set of objects
• which are all similar in terms of their
attributes
• A class consists of a class name, data
and methods.

Object Oriented Programming module | PDF page 3 | lines 35-45

፫. Encapsulation:
• means the wrapping up of data and
methods into a single unit (a class).
• It also means that the data inside a
class is hidden from everything
outside the class.
• The data can only be accessed by
invoking the methods of the class.
• It is placing the data and the
functions that work on that data in
the same place.

Question sources

Computer Science model exam 2 · Question 88

Question 141 | Q00748

Which OOP concept describes one operation in a class hierarchy producing different behavior for different runtime object types?

A. Aggregation

B. Polymorphism

C. Inheritance

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer B: Polymorphism

Polymorphism lets code using a common type invoke behavior appropriate to each concrete object. Eligible overriding supplies this effect at runtime: the same inherited operation may dispatch to a different subclass implementation. Inheritance establishes the relationship that permits those related types, but does not alone name the varying behavior. Aggregation instead models a whole grouping parts, rather than method dispatch.

Why each choice is right or wrong

A - Incorrect: Aggregation describes whole-part object relationships, not varying operation behavior.

B - Correct: Polymorphism names behavior selected appropriately for different concrete object types through a common operation.

C - Incorrect: Inheritance establishes the subtype hierarchy; polymorphism names the differing behavior through that hierarchy.

Primary note proof

Study-note section OOP15

> For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Study-note section OOP-S05

> Aggregation models one instance grouping other instances as a whole with parts. It is an object-relationship concept, not a method-dispatch rule. Inheritance instead relates a specialized class to a general class; overridden operations can then provide polymorphic behavior.

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 58-64

2. Polymorphism:
• The word polymorphism means the ability to take more than one form.
• In terms of the OOP, this means that a particular operation may behave differently for different sub-classes of the same class.

Object Oriented Programming module | PDF page 11 | lines 55-67

Method Overriding
In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, this is called method Overriding.
• the method in the subclass is said to override the method in the superclass.
• The Method in the super class is known as an overridden method which is called within a subclass.
• The version of the method defined by the superclass will be hidden.

Question sources

Computer Science model exam 2 · Question 89

Question 142 | Q00896

Which statement correctly describes the implicit object argument in an ordinary C++17 member-function call?

A. The this pointer is supplied as the implicit object argument to ordinary non-static member functions

B. The this pointer is supplied as a hidden argument to static member variables

C. The this pointer is supplied as a hidden argument to static member functions

D. The this pointer is supplied as a hidden argument to every member function, including static functions

Hint

Use the stated language rules and identify whether a member call has an implicit current object.

Answer A: The this pointer is supplied as the implicit object argument to ordinary non-static member functions

In ordinary C++17 non-static member functions, this points to the object on which the function is invoked and can be understood as the implicit object argument. Member variables do not receive call arguments. Static member functions have no current instance and no this pointer, so a claim about every member function is also too broad. Correcting variables to functions in A restores the intended distinction.

Why each choice is right or wrong

A - Correct: Ordinary non-static member functions have the implicit object represented by this.

B - Incorrect: Variables are not calls and do not receive hidden object arguments.

C - Incorrect: Static member functions have no this pointer.

D - Incorrect: Including static functions makes the universal claim false.

Primary note proof

Study-note section OOP17

> In ordinary C++17 non-static member functions, this is a pointer to the object on which the function is called; this->field selects a member. It is often explained as an implicit object argument to a member function call. It is not an argument passed to member variables, and a static member function has no this pointer.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 14 | lines 2-19

> Static methods and Final Variables
> • A static method can access only static data.
> • A static method can call only other static data (Static methods & variables), but can not call a non-static data from it.
> • Static method is belongs to the class and not to the object.
> • A static method can be accessed directly by the class name and doesn‟t need any object.
> • i.e ClassName.methodName(args)
> • A static method cannot refer to "this" or "super" keywords in anyway
> • But can be overloaded since they are resolved using static binding by compiler at compile time.

Question sources

Computer Science Exit Exam · 2018 · Question 9

Question 143 | Q00904

What does this Java program print?
public class Calculation {
public static void main(String[] args) {
int sum = 0;
for (int j = 1; j <= 10; j++) {
sum = sum + j;
}
System.out.println(+sum);
}
}

A. 55

B. 44

C. 33

D. 66

Hint

Trace condition tests, control transfers and printing as separate steps.

Answer A: 55

The loop adds each integer from 1 through 10 exactly once. Their sum is 10×11/2=55, and unary plus does not change that integer before printing. The malformed trailing token is replaced by the required closing braces, leaving the loop itself unchanged. Forty-four and thirty-three are not the inclusive sum, while sixty-six would correspond to additionally summing 11, beyond the bound.

Why each choice is right or wrong

A - Correct: The inclusive sum 1+2+...+10 is 55.

B - Incorrect: 44 is not the sum of the ten loop values.

C - Incorrect: 33 is not the sum produced by the inclusive loop.

D - Incorrect: 66 would include an extra 11 beyond the stated upper bound.

Primary note proof

Study-note section OOP25

> A loop summing j=1..10 yields 55.

Study-note section OOP-S03

> Prefix ++ and -- update a variable before supplying its value; postfix forms supply the old value before the update. Multiplication, division and remainder bind before addition and subtraction;

same-level binary arithmetic groups left to right. Integer division truncates toward zero. Unary plus preserves its numeric operand’s value.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 7 | lines 26-41

Iteration: The for(), while() and
do...while() statements
4. Jumping Statements: to jump out
of loops and to control other areas of
program java uses break and
continue statements.
The break statement: used to exit a
loop
The Continue Statement: used to
force program to go back to the top
of a loop
Labels: Java does not include a goto
statement.
Instead of goto, Java allows you to
combine break and continue with a
label.

Object Oriented Programming module | PDF page 4 | lines 21-34

✓ Operators: Are symbols that
take one or more arguments
(operands) and operates on
them to produce a result. An
operator performs a function
on one (Unary), two (Binary),
or three operands (Ternary).
Java has one ternary operator,
?:, which is a short-hand if-
else statement. E.g. c=x?w:s;
Operators can be Arithmetic,
Logical, assignment,
Relational, conditional,
Increment,

Question sources

Computer Science Exit Exam · 2018 · Question 17

Question 144 | Q00905

Which exact case-sensitive Java keyword declares a subclass relationship to a superclass?

A. Inherits

B. Superclass

C. extends

D. Implements

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer C: extends

A Java class declares its superclass with extends, as in class Dog extends Animal. The lowercase spelling is required because Java identifiers and keywords are case sensitive. implements is used for interface contracts rather than the superclass relationship, while Inherits and Superclass are not the required keyword. Correcting the option's initial letter makes the subclass-declaration question exact without changing its concept.

Why each choice is right or wrong

A - Incorrect: Inherits is not the Java subclass-declaration keyword.

B - Incorrect: Superclass names the relationship but is not its declaration keyword.

C - Correct: extends is the exact keyword used between subclass and superclass names.

D - Incorrect: implements concerns interfaces, and the printed capitalized word is not the required keyword.

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Study-note section OOP19

> interface Payable{double getPaymentAmount();} declares an abstract public operation. class Invoice implements Payable must provide public double getPaymentAmount(), perhaps returning getQuantity()*getPricePerItem(). A class can extend a superclass and implement zero or more interfaces. An interface can extend several interfaces.

Study-note section OOP28

> Java identifiers are case sensitive; $ is allowed but @ is not an identifier character.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

What is Inheritance?
• Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Question sources

Computer Science Exit Exam · 2018 · Question 18

Question 145 | Q00930

A BankAccount keeps its balance and deposit operation together. Clients cannot assign balance directly and must use validated operations. Which principle specifically names this bundling of state with operations and control of access to that state?

A. Abstraction

B. Inheritance

C. Polymorphism

D. Encapsulation

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer D: Encapsulation controls which information of an object is exposed.

The question specifically asks for the mechanism that groups an object's state with its operations and controls access to that state. That is encapsulation. Abstraction also matters in the account's public view, but it names the essential operations presented to callers rather than the requested state-and-operation boundary. Inheritance and polymorphism concern subtype relationships and behavior through common types, respectively.

Why each choice is right or wrong

A - Incorrect: Abstraction emphasizes the essential caller-facing view; it does not specifically name the requested grouping and state-access boundary.

B - Incorrect: Inheritance forms a subtype relationship rather than the requested state-and-operation boundary.

C - Incorrect: Polymorphism selects behavior through a common type rather than grouping and controlling this state.

D - Correct: Encapsulation specifically groups the balance with its operations and controls direct access.

Primary note proof

Study-note section OOP02

> Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller. These ideas overlap: a private balance plus deposit(amount) can demonstrate all three, depending on what the question emphasizes.

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 27-33

> 0. Abstraction:
> • provides only essential information
> to the outside world.
> • Or hiding their background details,
> i.e., to represent the needed
> information in program without
> presenting the details.

Object Oriented Programming module | PDF page 3 | lines 35-45

ვ. Encapsulation:
• means the wrapping up of data and methods into a single unit (a class).
• It also means that the data inside a class is hidden from everything outside the class.
• The data can only be accessed by invoking the methods of the class.
• It is placing the data and the functions that work on that data in the same place.

Question sources

Computer Science Exit Exam · 2018 · Question 43

Question 146 | Q00952

Consider the following code snippet
class Animal { void sound() { System.out.printIn("Animal sound"); } }
class Dog extends Animal { void sound() { System.out.printIn("Bark!"); } }
Which OOP concept is illustrated when Dog provides its own version of sound?

A. Method overriding

B. Method overloading

C. Abstraction

D. Encapsulation

Hint

Check inheritance, signature compatibility, visibility and eligibility for an override.

Answer A: Method overriding

Dog repeats sound() from Animal with its own body: that is a subclass method with the same name/signature as its superclass method. Literal printIn is a typo, but does not change the declaration relationship.

Why each choice is right or wrong

A - Correct: Matches the same-signature subclass relationship

B - Incorrect: No changed parameter list is shown

C - Incorrect: No abstract declaration or essential-interface example is needed

D - Incorrect: The example emphasizes replacement behavior, not grouping hidden data

Primary note proof

Study-note section OOP14

A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype.

Study-note section OOP12

Overloaded methods share a name but have different parameter types, counts or type order.

Study-note section OOP37

Check exact Java spelling and syntax: extends differs from Extends, println differs from printIn/printin, constructor names must match class names, and braces and operators such as <= and == affect validity and behavior. Java is case sensitive: an invalid program may not compile even when a possible trace seems clear.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 11 | lines 55-67

Method Overriding
In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, this is called method Overriding.
• the method in the subclass is said to override the method in the superclass.
• The Method in the super class is known as an overridden method which is called within a subclass.
• The version of the method defined by the superclass will be hidden.

Question sources

Computer Science Exit Exam · 2018 · Question 65

Question 147 | Q00956

Which most restrictive Java member-access category admits unrelated classes in the same package and qualified subclass access in other packages, while excluding unrelated classes in other packages?

A. Public

B. Default

C. Private

D. Protected

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer D: Protected, subject to cross-package receiver restrictions.

Protected combines same-package access with qualified subclass access across packages. It therefore reaches both required contexts while excluding unrelated other-package clients. Public is broader and violates that exclusion. Package/default access does not reach the other-package subclass, while private is narrower than the required same-package access. Cross-package instance access must still use the inherited context or an appropriate subclass-typed receiver.

Why each choice is right or wrong

A - Incorrect: Public also admits unrelated classes in other packages, violating the explicit exclusion.

B - Incorrect: Package/default access does not admit the required subclass access across packages.

C - Incorrect: Private does not grant access to unrelated same-package classes.

D - Correct: Protected grants the required same-package and qualified cross-package subclass access without granting universal access.

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible. Top-level classes/interfaces may be public or package-private, not protected/private. Member types have additional visibility possibilities.

Study-note section OOP08

> Across packages, a subclass accessing a protected instance member must do so through its own inherited context or an appropriate subclass-typed receiver, not any arbitrary superclass object.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

> Access Specifiers
> Access level modifiers determine whether
> other classes can use a particular field or
> invoke a particular method.
> Their are four types access levels:
> 1. No Modifier(Default): Visible to
> the package(source file), and it is
> called package-private
> 2. Private: Visible to the class only.
> 3. Public: Visible to the world.

4. Protected: Visible to the package
and all subclasses.

Question sources

Computer Science Exit Exam · 2018 · Question 69

Question 148 | Q00965

In abstract class Employee, calculateTax must remain a public concrete instance method while explicitly forbidding subclasses from overriding it. Which declaration achieves that requirement?

A. public abstract double calculateTax();

B. public final double calculateTax(){ return 0.0; }

C. private double calculateTax(){ return 0.0; }

D. public static double calculateTax(){ return 0.0; }

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer B: public final double calculateTax(){ return 0.0; }

final explicitly prohibits overriding an instance method while allowing it to remain public and concrete. abstract instead requires an implementation and supplies no body. private would remove public access and create independent child declarations rather than preserve the required public contract. static changes the method into a class operation whose same-signature child method hides it. Only the public final instance declaration satisfies all stated requirements.

Why each choice is right or wrong

A - Incorrect: abstract removes the concrete implementation and permits or requires overriding.

B - Correct: public final retains public concrete instance behavior and prohibits overriding.

C - Incorrect: private violates the required public accessibility.

D - Incorrect: static violates the requirement to remain an instance method.

Primary note proof

Study-note section OOP07

final prevents reassignment of a variable, overriding of a method or extension of a class.

Study-note section OOP14

A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype. Visibility cannot become more restrictive, and checked exceptions cannot be broadened incompatibly. @Override

requests compiler checking; Java has no override keyword. Constructors, final methods, static methods and private parent methods are not overridden.

Study-note section OOP18

An abstract method has no body; abstract and final conflict because an operation cannot require overriding while forbidding it.

Study-note section OOP08

For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Study-note section OOP16

Fields are hidden rather than dynamically overridden, and same-signature static methods hide rather than override.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 24 | lines 6-16

Using final with Inheritance
• The keyword final has three uses.
1. To create the equivalent of a named constant (final variable)
- A variable declared as final prevents the content of that variable being modified.
2. To Prevent Overriding in Methods (final method) - A method declared as final cannot be overridden.

Object Oriented Programming module | PDF page 21 | lines 26-39

Rules of method overriding in Java..
• Argument list: The argument list of overriding (method of subclass) and Overridden (method of super class) method must be similar.
• The data types of the arguments and their sequence should exactly match.
• Method names: must be similar.
• private, static and final methods cannot be overridden as they are local to the class.
• Access Modifier of the overriding method cannot be more restrictive than the overridden method of parent class.

Question sources

Computer Science Exit Exam · 2018 · Question 78

Question 149 | Q00987

What will be the output for the Java expression: 8 - 2 * 5 * (3 + 9 / 3) + 6 / 3

A. -150

B. -30

C. -50

D. 50

Hint

Respect operator grouping and operand types, then trace each operation in evaluation order.

Answer C: -50.

Inside the parentheses, 9/3=3, so the group is 3+3=6. The multiplication gives 2*5*6=60, and the remaining integer division gives 6/3=2. The result is 8-60+2=-50, matching OOP23's explicit example. The negative sign follows from subtracting the larger product.

Why each choice is right or wrong

A - Incorrect: -150 does not follow the parenthesized value 6 and product 60 obtained from the stated operands.

B - Incorrect: -30 miscalculates the product or remaining terms; 8-60+2 is -50.

C - Correct: The grouped arithmetic gives 8-60+2, which equals -50.

D - Incorrect: The product is subtracted, so changing -50 to positive 50 loses the required sign.

Primary note proof

Study-note section OOP23

> In 8-2*5*(3+9/3)+6/3, the result is -50.

Study-note section OOP-S03

> Prefix ++ and -- update a variable before supplying its value; postfix forms supply the old value before the update. Multiplication, division and remainder bind before addition and subtraction; same-level binary arithmetic groups left to right. Integer division truncates toward zero. Unary plus preserves its numeric operand's value.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that
> take one or more arguments

(operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Question sources

National Exit Exam practice questions · Question 1

Computer Science Exit Exam practice collection · Question 80

Question 150 | Q00988

Which one of the following is used for constants in Java?

A. const

B. final

C. define

D. All of the above

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer B: final

final creates the equivalent of a named constant by preventing value modification.

Why each choice is right or wrong

A - Incorrect: Not the keyword given for constants

B - Correct: Explicitly stated named-constant mechanism

C - Incorrect: Not Java’s named-constant keyword

D - Incorrect: Only final is established

Primary note proof

Study-note section OOP07

final prevents reassignment of a variable, overriding of a method or extension of a class. A final reference may still refer to a mutable object. A shared mathematical PI is a good static final candidate; a student name normally belongs to an instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 24 | lines 6-16

Using final with Inheritance
• The keyword final has three uses.
1. To create the equivalent of a named constant (final variable)
- A variable declared as final prevents the content of that variable being modified.
2. To Prevent Overriding in Methods (final method) - A method declared as final cannot be overridden.

Question sources

National Exit Exam practice questions · Question 2

Computer Science Exit Exam practice collection · Question 81

Question 151 | Q01172

In which character method returns the uppercase form of the specified char value.

A. a | toUpperCase()

B. i? | toLowerCase()

C. @) toString()

D. | isDigit()

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer A: Character.toUpperCase(ch).

OOP22 names Character.toUpperCase(ch) as the operation that converts a character to uppercase. A supplies the correct method name, with its argument abbreviated away. Lowercase conversion, string representation and a digit test have different purposes.

Why each choice is right or wrong

A - Correct: toUpperCase is the named conversion; an actual Character call supplies the char argument.

B - Incorrect: toLowerCase names the opposite letter-case conversion rather than producing uppercase.

C - Incorrect: toString names textual representation rather than uppercase conversion of the supplied character.

D - Incorrect: isDigit names a predicate about a character, not an uppercase conversion operation.

Primary note proof

Study-note section OOP-S04

> Character.toUpperCase(ch) supplies the uppercase equivalent where one exists; Character.toLowerCase(ch) supplies the lowercase equivalent. Character.isDigit(ch) tests whether a character is a digit and returns boolean. Character.toString(ch) supplies a one-character String rather than converting letter case.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 14-16

> 5. String Class:- Immutable
> Class created to manipulate
> strings(Sequence of chars).

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)
> 2. Reference types: Arrays, classes,
> and interfaces, String)
> • Note: Boolean type has a value true or
> false (there is no conversion b/n Boolean
> and other types).
> • All non-primitive types are reference
> types, so all class types are reference
> types.
> • All integers in Java are signed, Java
> doesn‟t support unsigned integers.

Question sources

Computer Science Exit Exam practice questions · Question 33

Question 152 | Q01173

In which parameter is the regular expression to which this string is to be matched?

A. a | regul x

B. Q « x

C. |@| regex v

D. | None of the above x

Hint

Separate identity, content equality, immutable values and references that may be reassigned.

Answer C: regex, in the documented Java String.matches(String regex) signature.

In the Java String-matching API, matches(String regex) names its regular-expression parameter regex. This answers the API-signature question; it does not make regex a mandatory parameter name in arbitrary programmer-defined methods.

Why each choice is right or wrong

A - Incorrect: The image recovers regul, which is not the parameter name in OOP22's documented matches signature.

B - Incorrect: The image recovers reg; the documented String.matches parameter is named regex, not reg.

C - Correct: The image places regex at C, matching the regular-expression parameter named in OOP22.

D - Incorrect: The documented parameter name is present at C, so none of the above is false.

Primary note proof

Study-note section OOP22

> charAt(index) returns a char; length() counts UTF-16 units; substring(begin,end) excludes end. Character.toUpperCase(ch) converts a character, and matches(String regex) takes a regular-expression parameter named regex.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 14-16

> 5. String Class:- Immutable
> Class created to manipulate
> strings(Sequence of chars).

Question sources

Computer Science Exit Exam practice questions · Question 34

Question 153 | Q01212

Which one of the following is not the purpose of the super keyword?

A. Refer to the current object

B. Call superclass constructor

C. Access hidden variables

D. Invoke superclass method with the same name as the subclass method

Hint

Distinguish the current instance, parent context, hidden fields and static dispatch.

Answer A: Refer to the current object

super refers to the immediate superclass and can access its members or constructor. It is distinguished from this, which refers to the current instance.

Why each choice is right or wrong

A - Correct: Current-object role belongs to this, not the superclass qualifier

B - Incorrect: Explicit superclass-constructor use

C - Incorrect: super.member can resolve hidden variables

D - Incorrect: Explicit superclass-method access use

Primary note proof

Study-note section OOP16

> Java this denotes the current object; this.name=name distinguishes a field from a parameter. super.name selects a hidden superclass field; super.sound() directly requests the parent implementation. It does not create a second parent object. An unqualified instance call may dispatch dynamically; an explicit super call is different.

Study-note section OOP10

> Under Java 17 rules, an explicit this(args) or super(args) invocation must be the first constructor statement. this invokes another constructor in the same class; super invokes the immediate superclass constructor. A constructor cannot explicitly invoke both.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 17 | lines 51-74

> The super keyword
> • Since encapsulation is a primary
> attribute of OOP, Whenever a subclass
> needs to refer to its immediate
> superclass, it can do so by use of the
> keyword super.

• Super has two general forms.
1. Used to access parent class
instance members.
2. Used to access parent class
constructor.
1. Used to access parent class
instance members.
• super acts somewhat like this, except
that it always refers to the superclass of
the subclass in which it is used.
• Super has the form: super.member Here,
member can be either a method or an
instance variable.
• Super is most applicable to resolve
member hiding.
• super( ) must always be the first
statement executed inside a subclass‟
constructor.

Question sources

Computer Science model exam 2 · Question 33

Question 154 | Q01219

Which one of the following has the same name as the class name?

A. Method

B. Variable

C. Constructor

D. Package name

Hint

Check the class name, return-type absence and the constructors actually declared.

Answer C: Constructor

Constructors have the same name as their class and no return type; a regular method need not have that name.

Why each choice is right or wrong

A - Incorrect: No requirement for ordinary methods to match the class name

B - Incorrect: Variables have their own identifiers

C - Correct: This naming rule applies to constructors

D - Incorrect: Packages are organizational names, not constructor names

Primary note proof

Study-note section OOP09

A constructor has the class's exact name and no return type, not even void. It initializes the new object and can be overloaded or private.

Study-note section OOP28

Packages organize names and control access; hierarchical spelling does not grant special access between a package and its subpackages.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 12 | lines 24-47

Constructors
• Performs automatic initialization of objects. .:.
• Initializes an object immediately upon creation.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
• The default constructor automatically initializes all instance variables to zero and all object references to NULL.
• The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
• Once you define your own constructor, the default constructor is no longer used.
• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Question sources

Computer Science model exam 2 · Question 40

Question 155 | Q01227

Which statement is true about the class concept?

A. The role of a class is to define the attributes and methods.

B. Class is used to distinguish one type of object from another.

C. A set of objects that share common methods, structure, and behavior.

D. All of the above

Hint

Distinguish an object’s stored state, operations, identity and class definition.

Answer D: All of the above

A class defines the common data/method shape of its objects and is itself a reference type. That supports attributes/methods, a distinct type, and a collection of similar objects.

Why each choice is right or wrong

A - Incorrect: Class consists of data and methods

B - Incorrect: Class defines a reference type and its objects' shape

C - Incorrect: Matches the definition of a set of similar objects

D - Correct: Combines the three supported descriptions

Primary note proof

Study-note section OOP01

> An object has identity, state and behavior. State is stored in fields; behavior is provided by methods. A Rectangle object may hold width and height and answer getArea() or getPerimeter(). Two objects can have equal fields and still be distinct objects. A class defines their common structure and behavior; an object is a particular instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 2-18

> • Are the basic runtime entities in
> an OO program.
> • Are identifiable things or
> individuals?
> • Are of significance to the system
> • Have states reflected in the
> attributes (data) held about
> the object
> • Have characteristic behaviour
> (i.e. functionality)
> ६. Class:

• Is a blueprint for an object.
• is the definition of a set of objects
• which are all similar in terms of their attributes
• A class consists of a class name, data and methods.

Object Oriented Programming module | PDF page 8 | lines 4-20

2.1. Fundamentals of Classes
• A class is a blue print from which individual objects are created.
• Class is a template for an object, and an object is an instance of a class
• Class is the logical construct upon which the entire Java language is built because it defines the shape and nature of an object.
• As such, the class forms the basis for object-oriented programming in Java.
• Classes usually consist of two things: instance variables and methods
• Any concept you wish to implement in a Java program must be encapsulated within a class.

Question sources

Computer Science model exam 2 · Question 48

Question 156 | Q01233

Which one is used to achieve the encapsulation concept in Java?

A. Declare the variables of a class as private.

B. Provide public setter and getter methods to modify and view the variable's values.

C. By using an Interface.

D. A & B

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer D: A & B

Hidden private fields plus accessible methods provide controlled access to class data. A alone hides data; B supplies the permitted interaction boundary.

Why each choice is right or wrong

A - Incorrect: Provides class-only data visibility

B - Incorrect: Provides the methods through which hidden data can be accessed

C - Incorrect: An interface alone does not hide a class's data fields

D - Correct: Combines hiding and controlled method access

Primary note proof

Study-note section OOP02

> Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface.

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Study-note section OOP11

> An accessor/getter returns information; a mutator/setter changes state. Encapsulation does not require blindly generating setters for every field: validate changes and expose only useful operations.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 35-45

> ፭. Encapsulation:
> • means the wrapping up of data and methods into a single unit (a class).
> • It also means that the data inside a class is hidden from everything outside the class.
> • The data can only be accessed by invoking the methods of the class.
> • It is placing the data and the functions that work on that data in the same place.

Object Oriented Programming module | PDF page 13 | lines 62-73

> Access Specifiers
> Access level modifiers determine whether

other classes can use a particular field or invoke a particular method.
Their are four types access levels:
1. No Modifier(Default): Visible to the package(source file), and it is called package-private
2. Private: Visible to the class only.
3. Public: Visible to the world.
4. Protected: Visible to the package and all subclasses.

Question sources

Computer Science model exam 2 · Question 54

Question 157 | Q01234

Which one of the following is the collection of statements that performs a specific operation?

A. Class

B. Package

C. Variable

D. Method

Hint

Separate parameters from arguments and operations that read state from operations that change it.

Answer D: Method

Methods are class members that perform operations and contain executable bodies.

Why each choice is right or wrong

A - Incorrect: A class groups fields and methods, not just one operation's statements

B - Incorrect: Package groups classes/interfaces

C - Incorrect: Variable stores data rather than an operation body

D - Correct: Matches an operation implemented by a statement body

Primary note proof

Study-note section OOP11

A method has modifiers, return type, name, parameter list and body: public int getAge(){return age;}. A void method returns no value. Parameters name the inputs in the declaration; arguments supply values at a call.

Study-note section OOP01

An object has identity, state and behavior. State is stored in fields; behavior is provided by methods. A Rectangle object may hold width and height and answer getArea() or getPerimeter().

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 10 | lines 62-72

Methods
• Are members of classes and do
operations.
• Allows different classes to
communicate each other.
• E.g. type name(parameter-
list)
{ // body of method
//....
return value; // if
type is not void

Question sources

Computer Science model exam 2 · Question 55

Question 158 | Q01255

For this valid Java instance method inside a class, which input/output statement is correct?

```
public void test(boolean a, boolean b) {
if (a) {
System.out.println("A");
} else if (a && b) {
System.out.println("A && B");
} else {
if (!b) {
System.out.println("!B");
} else {
System.out.println("None");
}
}
}
```

A. If a and b are both true, the output is "A && B".

B. If a is true and b is false, the output is "!B".

C. If a is false and b is true, the output is "None".

D. If a and b are both false, the output is "None".

Hint

Trace condition tests, control transfers and printing as separate steps.

Answer C: a=false and b=true prints None.

If a is true, the first branch prints A regardless of b. If a is false, the else-if condition a && b is necessarily false, so execution enters the final else. There !b selects !B when b is false and None when b is true. Only C pairs its input values with the resulting output.

Why each choice is right or wrong

A - Incorrect: When a is true the first branch prints A, not the unreachable A && B branch.

B - Incorrect: a=true selects A immediately regardless of b, so !B does not print.

C - Correct: a=false bypasses both earlier tests and b=true makes !b false, selecting None.

D - Incorrect: With both false, the final !b test is true and prints !B rather than None.

Primary note proof

Study-note section OOP25

> In if(a)... else if(a&&b)..., the second condition can never be true on a path reaching it because a is already false.

Study-note section OOP-S03

> For Boolean values, ! reverses true and false, && is true only when both operands are true, and || is true when at least one is true.

Study-note section OOP-S03

> An if selects its true branch or, when present, its else branch; only the selected branch executes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 7 | lines 48-54

> 2. Declaration statements: declares/
> Introduces a very first variables E.g.
> double c = 8.4;
> 3. Control flow statements: regulates
> the order of statements execution
> Selection: the if() and switch()
> statements

Object Oriented Programming module | PDF page 4 | lines 11-20

> ✓ Reserved keywords: class,
> extends, implements, for...
> ✓ Identifiers: programmers
> given name; grade, gRade...
> ✓ Literals (Constants values):

55, 60.78, “Hi”, „M“... Java language specifies five major types of literals => integer, floating, character, string & Boolean literals

Question sources

Computer Science model exam 2 · Question 76

Question 159 | Q01262

Which one of the following methods can be overriding?

A. Abstract method

B. Final Method

C. Private method

D. Static method

Hint

Check inheritance, signature compatibility, visibility and eligibility for an override.

Answer A: Abstract method

Subclasses provide implementations of inherited abstract methods; the overriding rules explicitly exclude private, static and final methods.

Why each choice is right or wrong

A - Correct: A child supplies the abstract method's implementation

B - Incorrect: Explicitly prohibited

C - Incorrect: Explicitly prohibited

D - Incorrect: Explicitly prohibited; static behavior is not overridden

Primary note proof

Study-note section OOP14

A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype.

Study-note section OOP14

Constructors, final methods, static methods and private parent methods are not overridden.

Study-note section OOP14

An abstract inherited method is implemented by an overriding method in a concrete subclass.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 21 | lines 26-39

Rules of method overriding in Java..
• Argument list: The argument list of overriding (method of subclass) and Overridden (method of super class) method must be similar.
• The data types of the arguments and their sequence should exactly match.
• Method names: must be similar.
• private, static and final methods cannot be overridden as they are local to the class.
• Access Modifier of the overriding method cannot be more restrictive than the overridden method of parent class.

Object Oriented Programming module | PDF page 23 | lines 57-81

Abstract Classes
• A classes declared with abstract keyword and can‟t be instantiated.
• Used to hide meaningless methods:
• There can be no objects of an abstract class. .:.
• An abstract class cannot be directly instantiated with the new operator.
• You cannot declare abstract constructors, or abstract static methods
• Syntax: abstract class class_name
Example:
abstract class A {
abstract void callme();
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
}}
class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
}}

Question sources

Computer Science model exam 2 · Question 83

Question 160 | Q01503

Considering the following program, Which one of the following statements gives syntax error?

```
class Shape {
double x;
void area() { ... }
}
class Circle extends Shape {
double PI = 3.14;
void display() { ... }
}
```

A. Shape s = new Circle(); s.area();

B. Shape s = new Shape(); s.area();

C. Shape s = new Circle(); s.display();

D. Circle c = new Circle(); c.area();

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer C: Shape s = new Circle(); s.display();

A superclass reference can refer to a child object, but only exposes members available through its declared type. An Animal reference cannot directly access the Dog-specific field breed. Shape has no display(), so s.display() is the analogous compile-time member error.

Why each choice is right or wrong

A - Incorrect: Shape reference can call its declared area; runtime dispatch applies

B - Incorrect: Shape object and its declared area are consistent

C - Correct: display is only declared in Circle, not the reference type Shape

D - Incorrect: Circle inherits area from Shape

Primary note proof

Study-note section OOP15

> For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile. If fetch() exists only in Dog, a.fetch() fails compilation. For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 17 | lines 34-49

A Superclass Variable Can Reference a Subclass Object
class Animal{
string name; int age
int show(){...}}
class Dog extends Animal{
String breed;
String barking(){
}}
//main{
Animal wild=new Animal();
Dog fries=new Dog();
wild=fries;
Wild.show()// correct show() is defined in Animal
Wild.breed="Woo woo"// ERROR

Object Oriented Programming module | PDF page 22 | lines 37-53

• Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
• Dynamic method dispatch is important because this is how Java implements run-time polymorphism
• When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs.
• Thus, this determination is made at run time.
• Method Overriding is an example of runtime polymorphism.

Question sources

Mock exam 1 · Question 19

Question 161 | Q01504

One of the following does not describe an object

A. Delegates responsibilities to other objects

B. It receives and sends message from/to other objects

C. It is an instance of a class

D. It is an abstract description of the state and behavior of a collection of similar objects

Hint

Distinguish an object's stored state, operations, identity and class definition.

Answer D: It is an abstract description of the state and behavior of a collection of similar objects

An object is a concrete runtime instance with state and behavior, whereas the abstract description of similar objects is the class blueprint.

Why each choice is right or wrong

A - Incorrect: Can describe collaborating operations, though delegation is not separately defined

B - Incorrect: Method communication is consistent with the class operations description

C - Incorrect: Explicit object definition

D - Correct: Describes the class template rather than one instance

Primary note proof

Study-note section OOP01

> An object has identity, state and behavior. State is stored in fields; behavior is provided by methods. A Rectangle object may hold width and height and answer getArea() or getPerimeter(). Two objects can have equal fields and still be distinct objects. A class defines their common structure and behavior; an object is a particular instance.

Study-note section OOP01

> Objects send messages through method calls such as obj.add(). They can delegate work to other objects; saying that objects cannot manage their own state is misleading.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 2-18

> • Are the basic runtime entities in
> an OO program.
> • Are identifiable things or
> individuals?
> • Are of significance to the system
> • Have states reflected in the

attributes (data) held about
the object
• Have characteristic behaviour
(i.e. functionality)
ह. Class:
• Is a blueprint for an object.
• is the definition of a set of objects
• which are all similar in terms of their
attributes
• A class consists of a class name, data
and methods.

Object Oriented Programming module | PDF page 10 | lines 62-72

Methods
• Are members of classes and do
operations.
• Allows different classes to
communicate each other.
• E.g. type name(parameter-
list)
{ // body of method
//....
return value; // if
type is not void

Object Oriented Programming module | PDF page 8 | lines 4-20

2.1. Fundamentals of Classes
• A class is a blue print from which
individual objects are created.
• Class is a template for an object, and
an object is an instance of a class
• Class is the logical construct upon
which the entire Java language is
built because it defines the shape and
nature of an object.
• As such, the class forms the basis for
object-oriented programming in
Java.
• Classes usually consist of two things:
instance variables and methods
• Any concept you wish to implement
in a Java program must be
encapsulated within a class.

Question sources

Mock exam 1 · Question 20

Question 162 | Q01505

Which statements about Java 17 constructors are false? Select all that apply.

A. A constructor can be overloaded

B. It has the same name as the class

C. can be invoked within any method without object instantiation.

D. the first line of the constructor must be a call on another constructor in the same class

E. Has no return statement

Hint

Check the class name, return-type absence and the constructors actually declared.

Answer C, D, E: can be invoked within any method without object instantiation.; the first line of the constructor must be a call on another constructor in the same class; Has no return statement

Constructors can be overloaded and must use the class's exact name, so A and B are true. They are not ordinary methods that can be called anywhere without construction. A same-class this(...) call is not mandatory; delegation may use super(...) or implicit super(). Constructors also allow bare return; even though they have no return type and cannot return an expression. Therefore C, D and E are false.

Why each choice is right or wrong

A - Incorrect: Overloading constructors is permitted, so this statement is true.

B - Incorrect: A constructor has the exact class name, so this statement is true.

C - Correct: Constructors are not ordinary methods callable from any method without construction.

D - Correct: An explicit same-class constructor call is not mandatory.

E - Correct: A bare return; is allowed, so the claim of no return statement is false.

Primary note proof

Study-note section OOP09

> A constructor has the class's exact name and no return type, not even void. It initializes the new object and can be overloaded or private. A private constructor can support a factory or controlled singleton creation; it does not mean the class can never create an object internally.

Study-note section OOP09

> Constructors are not inherited or overridden. They may contain return; but cannot return an expression. Fields get their default values before ordinary constructor initialization, rather than receiving defaults only because a default constructor was generated.

Study-note section OOP10

Under Java 17 rules, an explicit this(args) or super(args) invocation must be the first constructor statement. this invokes another constructor in the same class; super invokes the immediate superclass constructor. A constructor cannot explicitly invoke both. If there is no explicit delegation, an implicit super() is inserted; a matching accessible no-argument parent constructor must exist.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 12 | lines 24-47

Constructors
• Performs automatic initialization of objects. .:.
• Initializes an object immediately upon creation.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
• The default constructor automatically initializes all instance variables to zero and all object references to NULL.
• The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
• Once you define your own constructor, the default constructor is no longer used.
• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Object Oriented Programming module | PDF page 18 | lines 36-47

2. Used to access parent class constructor.
• A subclass can call a constructor defined by its superclass by use of the following form of super:
super(arg-list);
• Here, arg-list specifies any arguments needed by the constructor in the superclass.
• super( ) must always be the first

statement executed inside a subclass‟
constructor.

Question sources

Mock exam 1 · Question 21

Question 163 | Q01585

What is the output of the following program? Assume every necessary part of the program is included.

```
class Car {
void drive() {
System.out.print("generic noise:");
}
}
public class Casting {
public static void main(String []args){
Car a = new Car();
Ferrari d = (Ferrari)a;
a.drive();
d.brake();
}
}
class Ferrari extends Car {
void drive() {
System.out.print("drive:");
}
void brake() {
System.out.println("brake hard:");
}
}
```

A. Generic noise: brake hard

B. drive: brake hard

C. generic nose: drive: brake hard

D. Compiler Error

E. Run time Error

Hint

Separate compile-time cast compatibility from the object’s actual runtime type.

Answer E: Runtime ClassCastException before any drive or brake output.

a refers to an actual Car created with new Car(), not to a Ferrari. The cast is type-checkable because Ferrari is a subclass, but it fails at runtime as OOP21 states. Execution stops at the cast with ClassCastException, before a.drive() or d.brake() can print any of their messages.

Why each choice is right or wrong

A - Incorrect: The failing cast occurs before either drive or brake, so neither listed message prints.

B - Incorrect: No Ferrari object exists here; the cast throws before any dynamic drive or brake call.

C - Incorrect: The program never reaches any of the three proposed messages because the cast fails first.

D - Incorrect: The related-class cast can pass compile-time checking; the actual object type causes a runtime failure.

E - Correct: Casting a plain Car object to Ferrari throws ClassCastException at runtime before any method output.

Primary note proof

Study-note section OOP21

> Upcasting a subclass reference to a superclass is implicit and does not change the object. A downcast may pass compile-time type checks yet fail at runtime. Car a=new Car(); Ferrari f=(Ferrari)a; throws ClassCastException before subsequent drive() or brake() calls. It does not turn a plain Car into a Ferrari.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 26 | lines 32-42

> The instance of operator
> • In Java, instanceof operator is used to
> check the type of an object at runtime.
> • It is the means by which your program
> can obtain run-time type information
> about an object.
> • instanceof operator is also important in
> case of casting object at runtime.
> • instanceof operator return boolean value,
> if an object reference is of specified type
> then it return true otherwise false.

Object Oriented Programming module | PDF page 26 | lines 48-69

> E.g. of downcasting with instanceof
> operator
> class Parent{ }
> public class Child extends

```
Parent{
public void check(){
System.out.println("Sucessfull
Casting");
}
public static void
show(Parent p){
if(p instanceof
Child){
Child
b1=(Child)p;
b1.check();
}}
public static void
main(String[] args){
Parent p=new Child();
Child.show(p);
}} Output: Sucessfull
```

Object Oriented Programming module | PDF page 17 | lines 34-49

```
A Superclass Variable Can Reference a
Subclass Object
class Animal{
string name; int age
int show(){...}}
class Dog extends Animal{
String breed;
String barking(){
}}
//main{
Animal wild=new Animal();
Dog fries=new Dog();
wild=fries;
Wild.show()// correct show() is
defined in Animal
Wild.breed="Woo woo"// ERROR
```

Question sources

Mock exam 3 · Question 17

Question 164 | Q01624

Which Java local declarations are valid when considered independently? Assume no type named x exists and the references are not read before assignment. Select all that apply.

A. None of the choices is correct

B. int [ ] x, y = new int[5];

C. int x[ ] = new x[5];

D. int [ ] x = new int[4];

E. int x = {5, 4, 9};

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer B, D: int [ ] x, y = new int[5];; int [ ] x = new int[4];

In B, int[] is the shared type of both x and y. Only y receives a new five-element array, which is a legal declaration as long as unassigned x is not read. D separately creates a four-element int array. C cannot use the variable-like name x as an allocation type under the stated scope, and E declares a scalar int with an array initializer. Thus B and D are valid.

Why each choice is right or wrong

A - Incorrect: B and D are valid declarations, so none is incorrect.

B - Correct: Both references have type int[]; initializing only y is legal.

C - Incorrect: No type x exists, so new x[5] is not a valid allocation here.

D - Correct: This is a valid int[] declaration and allocation.

E - Incorrect: A scalar int cannot be initialized with a multi-element brace array initializer.

Primary note proof

Study-note section OOP26

> int[] a=new int[5]; and int a[]={1,2,3}; are valid. int[] a={}; creates a zero-length array; explicit new is unnecessary in a declaration initializer. int a[],b; makes only a an array; int[] c,d; makes both arrays. Array objects have fixed length after creation, although the reference may be reassigned.

Study-note section OOP05

> Locals occur in methods, constructors or blocks; they must be definitely assigned before use and cannot have public/private/protected modifiers.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java
(byte, short, int, long, float, double,
char & Boolean)
2. Reference types: Arrays, classes,
and interfaces, String)
• Note: Boolean type has a value true or
false (there is no conversion b/n Boolean
and other types).
• All non-primitive types are reference
types, so all class types are reference
types.
• All integers in Java are signed, Java
doesn‟t support unsigned integers.

Question sources

Mock exam 2 · Question 13

Question 165 | Q01625

What is a Java static initializer block?

A. A block that initializes class-level state during class initialization, before that class's first relevant active use, including its main invocation

B. None of the choices is correct.

C. There is no such block.

D. It is used to create synchronized code.

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer A: A block that initializes class-level state during class initialization, before that class's first relevant active use, including its main invocation

A static initializer executes during class initialization and can initialize static state. It runs before the class's first relevant active use, including invoking its main method, rather than simply whenever a class file is loaded. It is a real Java construct, so none and the denial option are false. Synchronization is a different concern and does not define a static initializer's purpose.

Why each choice is right or wrong

A - Correct: This states the class-state initialization purpose and correct initialization timing.

B - Incorrect: A correctly describes a real construct, so none is incorrect.

C - Incorrect: Static initializer blocks exist in Java.

D - Incorrect: Creating synchronized code is not the defining role of a static initializer.

Primary note proof

Study-note section OOP07

A static initializer runs during class initialization, before the class's first relevant active use, not simply whenever a file is loaded.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 5 | lines 39-61

C. Class/Static variables
• Are declared with the static keyword in a class, but outside a method, constructor or a block.
• There would only be one copy of each class variable per class, regardless of how many objects are created from it.
• Are rarely used other than being declared as constants. (public static final)
• Are stored in the static memory.
• Static variables are created when the program starts and destroyed when the program stops.
• Visibility is similar to instance variables. However, most static variables are declared public
• Default values are same as instance variables.
• Static variables can be accessed by calling with the class name ClassName.VariableName.

Object Oriented Programming module | PDF page 14 | lines 2-19

Static methods and Final Variables
• A static method can access only static data.
• A static method can call only other static data (Static methods & variables), but can not call a non-static data from it.
• Static method is belongs to the class and not to the object.
• A static method can be accessed directly by the class name and doesn‟t need any object.

• i.e ClassName.methodName(args)
• A static method cannot refer to "this" or "super" keywords in anyway
• But can be overloaded since they are resolved using static binding by compiler at compile time.

Question sources

Mock exam 2 · Question 14

Question 166 | Q01626

What is the output of the below given Java code? Assume every necessary components of the program are included.

```
int i;
for (i = 0; i < 5; i++) {
if (i % 2 == 0)
System.out.print ( i + " ");
System.out.print ( i + 1);
}
```

A. 0 12 34 5

B. None of the choices is correct

C. 0 2 4 6

D. 0 122 344 5

Hint

Trace condition tests, control transfers and printing as separate steps.

Answer D: 0 122 344 5.

Only the first print is controlled by if. At i=0,2,4 it prints the even value followed by a space; every iteration then prints i+1 without a separator. The contributions are "0 1", "2", "2 3", "4", "4 5". Joining them produces "0 122 344 5", exactly as OOP25 records.

Why each choice is right or wrong

A - Incorrect: This omits digits from the unconditional i+1 prints on odd and even iterations.

B - Incorrect: D supplies the exact concatenated output, so none of the choices is false.

C - Incorrect: The unconditional print runs five times; the output is not just an even-number sequence.

D - Correct: Joining every conditional and unconditional print yields the exact string 0 122 344 5.

Primary note proof

Study-note section OOP25

Count executions independently from printing. For i=0..4, if(i%2==0) print(i+" "); followed unconditionally by print(i+1); produces "0 122 344 5".

Study-note section OOP-S02

A print call emits its value without inserting a separator or ending the line. A println call emits its value followed by a line terminator. Spaces must come from the supplied text or explicit output operations; adjacent calls do not automatically separate their values.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 7 | lines 26-41

Iteration: The for(), while() and
do...while() statements
4. Jumping Statements: to jump out
of loops and to control other areas of
program java uses break and
continue statements.
The break statement: used to exit a
loop
The Continue Statement: used to
force program to go back to the top
of a loop
Labels: Java does not include a goto
statement.
Instead of goto, Java allows you to
combine break and continue with a
label.

Object Oriented Programming module | PDF page 4 | lines 21-34

✓ Operators: Are symbols that
take one or more arguments
(operands) and operates on
them to produce a result. An
operator performs a function
on one (Unary), two (Binary),
or three operands (Ternary).
Java has one ternary operator,
?:, which is a short-hand if-
else statement. E.g. c=x?w:s;
Operators can be Arithmetic,
Logical, assignment,
Relational, conditional,
Increment,

Question sources

Mock exam 2 · Question 15

Question 167 | Q00513B

_____ is a way of saying: This object is a type of that object.

A. IS-A

B. HAS-A

C. ARE-A

D. HAD-A

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer A: IS-A

IS-A describes type membership: an object belongs to a type or one of its subtypes.

Why each choice is right or wrong

A - Correct: Matches the named IS-A relationship

B - Incorrect: HAS-A would describe composition/containment rather than type membership

C - Incorrect: Not the named relationship

D - Incorrect: Not the named relationship

Primary note proof

Study-note section OOP13

> Use inheritance when a subtype can satisfy the parent's contract: Dog is an Animal. For shared Pet.name and speak(), put common state/contract in Pet, specialized fetch() in Dog, and appropriate speak() implementations in Dog and Cat. Composition represents has-a, such as Car containing Engine. Reuse alone does not justify an invalid is-a relationship.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 25 | lines 17-37

> • Polymorphism is the ability of an object to take on many forms.
> • The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
> • Any Java object that can pass more than

one IS-A test is considered to be polymorphic.
• In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.
• Polymorphism is derived from 2 Greek words: poly and morphs.
• The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
• polymorphism is the capability of an action or method to do different things based on the object that it is acting upon.

Object Oriented Programming module | PDF page 30 | lines 37-59

Chapter - Five: Java Interface
• An interface in java is a blueprint of a class.
• It has static constants and abstract methods.
• The interface in java is a mechanism to achieve abstraction.
• There can be only abstract methods in the java interface not method body (contain only signature, not body).
• It is used to achieve abstraction and multiple inheritance in Java.
• Java Interface also represents IS-A relationship.
• It cannot be instantiated just like abstract class.
• There are mainly three reasons to use interface.
• It is used to achieve abstraction.
• By interface, we can support the functionality of multiple inheritance.

Question sources

Object-Oriented Programming practice questions · Question 83

Question 168 | SM0012

Which Java member access modifier permits subclasses in other packages to access inherited members through their own inherited context, while excluding unrelated clients in those other packages? Same-package access is allowed.

A. protected

B. private

C. default

D. public

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer A: protected

protected includes same-package access and also permits qualified subclass access across packages. That matches the revised boundary, which excludes unrelated clients in other packages while allowing same-package clients. private does not generally expose inherited members to separately declared subclasses, package access does not reach other packages, and public allows unrelated external clients. The question no longer demands an impossible universal subclasses-only rule.

Why each choice is right or wrong

A - Correct: protected provides same-package access plus the specified qualified cross-package subclass access.

B - Incorrect: private does not grant the requested general cross-package subclass access.

C - Incorrect: Package/default access does not extend to subclasses in another package.

D - Incorrect: public also permits unrelated accessible external clients, violating the boundary.

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Study-note section OOP08

> Across packages, a subclass accessing a protected instance member must do so through its own inherited context or an appropriate subclass-typed receiver, not any arbitrary superclass object. protected therefore does not mean “subclasses only”: unrelated classes in the same package also have access.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

Access Specifiers
Access level modifiers determine whether other classes can use a particular field or invoke a particular method.
Their are four types access levels:
1. No Modifier(Default): Visible to the package(source file), and it is called package-private
2. Private: Visible to the class only.
3. Public: Visible to the world.
4. Protected: Visible to the package and all subclasses.

Question sources

MOE Computer Science Exit Exam · Question 3-4

National Computer Science Exit Exam · Question 63

National Computer Science Exit Exam · Question 66

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 97

Question 169 | SM0019

Which one of the following is the correct way of creating a String object?

A. String m = 'Computer Science';

B. String m = new String('Technology');

C. String m = 'H', 'e', 'l', 'l', 'o';

D. String m = new String("Exit Exam");

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer D: String m = new String("Exit Exam");

A String literal uses double quotes, while a character literal supplies one char in single quotes. D passes a valid String literal to the String constructor and assigns the resulting object reference. A and B instead use invalid multi-character character literals. C is not one valid String initializer: separate comma-delimited character literals do not form a String expression or a character-array constructor argument.

Why each choice is right or wrong

A - Incorrect: The multi-character text in single quotes is not a valid Java character literal or String expression.

B - Incorrect: new String does not make its invalid single-quoted multi-character argument legal.

C - Incorrect: The comma-separated character literals do not form one String initializer or a supplied char-array value.

D - Correct: A double-quoted String argument matches the String constructor and initializes the String reference.

Primary note proof

Study-note section OOP-S04

> A double-quoted string literal denotes a String object. new String(existingString) creates a String with the same text; new String(charArray) copies the array's characters into an immutable String. A single-quoted literal denotes one char, so several characters enclosed together in single quotes cannot supply a String.

Study-note section OOP03

> Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 14-16

> 5. String Class:- Immutable
> Class created to manipulate
> strings(Sequence of chars).

Object Oriented Programming module | PDF page 4 | lines 11-20

> ✓ Reserved keywords: class,
> extends, implements, for…
> ✓ Identifiers: programmers
> given name; grade, gRade…
> ✓ Literals (Constants values):
> 55, 60.78, "Hi", „M"… Java
> language specifies five major
> types of literals => integer,
> floating, character, string &
> Boolean literals

Object Oriented Programming module | PDF page 8 | lines 43-55

> • a class declaration only creates a
> template; it does not create an actual
> object.

• To actually create a Dog object, use the second statement.
• Dog myDog; // Creates only the template
• myDog = new Dog(); // create a Dog object called myDog
• After this statement executes, myDog will be an instance of Dog.

Question sources

MOE Computer Science Exit Exam · Question 5-3

National Computer Science Exit Exam · Question 68

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 98

Question 170 | SM0069

Which one of the following is false about String in Java?

A. In Java a string is treated as object

B. We can create a String object using String literal

C. We can create a String object using array of characters

D. Content of a string can be changed once the string is created

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer D: A created String's content cannot be changed.

OOP22 explicitly states that Java String objects are immutable. A variable can later reference a new String result, but the existing object's content does not change. D therefore contradicts the rule. The other choices describe String objects or ways of supplying initial content, which do not assert mutation of an already-created String.

Why each choice is right or wrong

A - Incorrect: String is an object type in the note; this is not the false mutability claim.

B - Incorrect: A String literal supplies String content; it does not imply the resulting object can later mutate.

C - Incorrect: Supplying initial character content creates a String; it does not change an existing String's immutable content.

D - Correct: This is false: Java String objects are immutable after creation, even if a variable is rebound.

Primary note proof

Study-note section OOP-S04

A double-quoted string literal denotes a String object. new String(existingString) creates a String with the same text; new String(charArray) copies the array's characters into an immutable String. A single-quoted literal denotes one char, so several characters enclosed together in single quotes cannot supply a String.

Study-note section OOP22

Java String objects are immutable. text += " World" assigns a reference to a new concatenated result and can print Hello World without mutating the original String object. final class String prevents subclassing; final alone does not make arbitrary objects immutable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 14-16

5. String Class:- Immutable
Class created to manipulate
strings(Sequence of chars).

Object Oriented Programming module | PDF page 8 | lines 43-55

• a class declaration only creates a
template; it does not create an actual
object.
• To actually create a Dog object, use
the second statement.
• Dog myDog; // Creates only
the template
• myDog = new Dog(); //
create a Dog object called
myDog
• After this statement executes,
myDog will be an instance of
Dog.

Object Oriented Programming module | PDF page 4 | lines 11-20

✓ Reserved keywords: class,
extends, implements, for…
✓ Identifiers: programmers
given name; grade, gRade…
✓ Literals (Constants values):

55, 60.78, “Hi”, „M“... Java language specifies five major types of literals => integer, floating, character, string & Boolean literals

Question sources

MOE Computer Science Exit Exam · Question 20-1

National Computer Science Exit Exam · Question 67

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 75

Question 171 | SM0074

Which concept refers to extending specialized classes from generalized classes in object-oriented programming?

A. Inheritance

B. Overriding

C. Overloading

D. Data field Encapsulation

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer A: Inheritance

A subclass is a specialized version of a superclass, and inheritance creates that relation.

Why each choice is right or wrong

A - Correct: Matches specialized-from-general hierarchy

B - Incorrect: Replaces an inherited method, not the class relationship

C - Incorrect: Adds same-name methods with changed parameters

D - Incorrect: Groups/controls data, not class specialization

Primary note proof

Study-note section OOP13

class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Study-note section OOP13

Use inheritance when a subtype can satisfy the parent's contract: Dog is an Animal.

Study-note section OOP02

Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

What is Inheritance?
• Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Question sources

MOE Computer Science Exit Exam · Question 21-2

National Computer Science Exit Exam · Question 47

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 99

Question 172 | SM0218

One of the following is NOT TRUE about inheritance:

A. The subclass can add variables, but cannot redefine them

B. The derived class (subclass) is allowed to add methods or redefine them

C. A feature that allows us to derive a class from another class

D. Allows programmers to customize without actually modifying the original class (superclass)

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer A: The subclass can add variables, but cannot redefine them

B declares its own i, hiding A.i, and adds its own methods. Thus subclass fields can be declared with an ancestor's name; A's cannot claim is false in the intended hiding sense.

Why each choice is right or wrong

A - Correct: Contradicted by B's int i hiding A.i

B - Incorrect: Subclass can add methods and override inherited ones

C - Incorrect: Matches the inheritance definition

D - Incorrect: Subclass extension permits customization without changing the parent

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent. A class extends one direct class and can implement multiple interfaces. Single, multilevel and hierarchical inheritance mean one parent, a chain, and several children of one parent.

Study-note section OOP16

> Fields are hidden rather than dynamically overridden, and same-signature static methods hide rather than override.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 18 | lines 14-29

```
// Using super to overcome name
hiding.
class A {
int i; }
class B extends A {
int i; // this i hides the i in
A
B(int a, int b) {
super.i = a; // i in A
i = b; // i in B }
void show() {
System.out.println("i in
superclass: " + super.i);
System.out.println("i in
subclass: " + i);
}}
```

Object Oriented Programming module | PDF page 15 | lines 23-53

What is Inheritance?
• Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{

```
.....}
class Sub extends Super{ }
```

Question sources

Computer Science Exit Exam · Sene 2016 · Question 6-2

Computer Science Exit Exam practice collection · Question 106

Question 173 | SM0237

Which OOP concept allows objects of different classes to be treated as objects of a common superclass?

A. Polymorphism

B. Inheritance

C. Abstraction

D. Encapsulation

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer A: Polymorphism

A parent-class reference referring to child objects is a common use of polymorphism. Inheritance establishes the relation, while polymorphism is the treatment through the common type.

Why each choice is right or wrong

A - Correct: Names the common-superclass many-forms treatment

B - Incorrect: Prerequisite relationship but not the treatment asked for

C - Incorrect: Hides background details

D - Incorrect: Groups data and methods

Primary note proof

Study-note section OOP15

> To solve dispatch questions: check the declaration and visibility, determine the compile-time signature, then identify the object's runtime class and its most specific override. An array Animal[]{new Dog(),new Cat()} can call speak() on each using one loop. New subclasses can supply behavior without rewriting this generic loop, which is the practical meaning of extensibility.

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 25 | lines 17-37

• Polymorphism is the ability of an object to take on many forms.
• The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
• Any Java object that can pass more than one IS-A test is considered to be polymorphic.
• In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.
• Polymorphism is derived from 2 Greek words: poly and morphs.
• The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
• polymorphism is the capability of an action or method to do different things based on the object that it is acting upon.

Object Oriented Programming module | PDF page 17 | lines 34-49

A Superclass Variable Can Reference a Subclass Object
class Animal{
string name; int age
int show(){...}}
class Dog extends Animal{
String breed;
String barking(){
}}
//main{
Animal wild=new Animal();
Dog fries=new Dog();
wild=fries;
Wild.show()// correct show() is defined in Animal
Wild.breed=“Woo woo”// ERROR

Question sources

Computer Science Exit Exam · Sene 2016 · Question 12-2

Question 174 | SM0243

Which purposes apply to a Java interface viewed as a capability contract? “Blueprint” means an operation contract for implementing classes, and “multiple inheritance” means multiple type inheritance rather than multiple superclass state inheritance. Select all that apply.

A. To define a blueprint for a class

B. To allow multiple inheritance

C. To create objects directly

D. To hide internal implementation details

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer A, B, D: To define a blueprint for a class; To allow multiple inheritance; To hide internal implementation details

An interface specifies an operation contract that implementing classes satisfy, so the qualified blueprint description is valid. A class can implement multiple interfaces, gaining multiple type inheritance. Clients can use the contract without depending on a particular implementation’s internals, supporting abstraction. Interfaces are not directly instantiated, so creating objects directly is not their purpose. The multi-select form therefore accepts A, B and D.

Why each choice is right or wrong

A - Correct: An operation blueprint is the contract that implementations must satisfy.

B - Correct: Implementing multiple interfaces supports multiple type inheritance.

C - Incorrect: An interface cannot be instantiated directly.

D - Correct: The public contract lets callers use capabilities without depending on internal implementation details.

Primary note proof

Study-note section OOP19

> interface Payable{double getPaymentAmount();} declares an abstract public operation. class Invoice implements Payable must provide public double getPaymentAmount(), perhaps returning getQuantity()*getPricePerItem().

Study-note section OOP02

> Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Study-note section OOP20

> Use an abstract class for shared state, construction and common implementation; use an interface for a capability across unrelated classes and multiple type inheritance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 30 | lines 37-59

Chapter - Five: Java Interface
• An interface in java is a blueprint of a class.
• It has static constants and abstract methods.
• The interface in java is a mechanism to achieve abstraction.
• There can be only abstract methods in the java interface not method body (contain only signature, not body).
• It is used to achieve abstraction and multiple inheritance in Java.
• Java Interface also represents IS-A relationship.
• It cannot be instantiated just like abstract class.
• There are mainly three reasons to use interface.
• It is used to achieve abstraction.
• By interface, we can support the functionality of multiple inheritance.

Object Oriented Programming module | PDF page 32 | lines 12-22

Extending multiple Interface
Multiple inheritance in Java by interface
• If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.
Extending multiple Interface
• A single interface can extend multiple interfaces, below is a simple example.

Variant 1 | Alternative version 1

Which purposes apply to Java interfaces as capability contracts, rather than complete per-instance state definitions? “Blueprint” means an operation contract, and “multiple inheritance” means multiple type inheritance. Select all that apply.

A. To create multiple constructors for a class

B. To define the state and behavior of an object

C. To define a blueprint of a class

D. To achieve multiple inheritance

Answer C, D: Interfaces define capability contracts and support multiple type inheritance.

This variant places the contract/blueprint purpose at C and multiple type inheritance at D. Both are valid: implementing objects satisfy the public operation contract, and a class may implement several interfaces. Interfaces have no constructors or per-instance fields, so they do not create class constructors or define an object’s complete state and implementation. Its selected labels therefore differ from the primary option set.

Why each choice is right or wrong

A - Incorrect: Interfaces have no constructors, so creating multiple constructors is not their purpose.

B - Incorrect: An interface specifies capabilities rather than complete per-instance state and implementation.

C - Correct: The defined blueprint is an operation contract for implementing classes.

D - Correct: Several interface types provide multiple type inheritance.

Primary note proof

Study-note section OOP19

> interface Payable{double getPaymentAmount();} declares an abstract public operation. class Invoice implements Payable must provide public double getPaymentAmount(), perhaps returning getQuantity()*getPricePerItem(). A class can extend a superclass and implement zero or more interfaces. An interface can extend several interfaces. No interface instance fields or constructors exist; fields are implicitly public static final.

Study-note section OOP20

> Use an abstract class for shared state, construction and common implementation; use an interface for a capability across unrelated classes and multiple type inheritance.

Related reading (context, not primary note proof)

Study-note section OOP02

Question sources

Computer Science Exit Exam · Sene 2016 · Question 14-2

Computer Science MCQ practice collection · Question 340

Computer Science Exit Exam practice collection · Question 104

Question 175 | SM0245

Which OOP relationship is formed when a Java class implements several interface contracts? Choose the relationship concept, rather than the language construct declaring each contract.

A. Polymorphism

B. Encapsulation

C. Inheritance

D. Interface

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer C: Inheritance, specifically multiple interface/type inheritance.

Implementing several interfaces gives the class several interface types and forms multiple type inheritance. Inheritance is therefore the relationship concept requested. Interface names a construct used to declare a contract, rather than the relationship itself. Polymorphism can use the resulting common types to select behavior, while encapsulation groups state and controls access. Neither names the requested multiple-type relationship.

Why each choice is right or wrong

A - Incorrect: Polymorphism uses common types for appropriate behavior; the question asks for the type relationship itself.

B - Incorrect: Encapsulation groups state and controls access rather than forming the specified type relationship.

C - Correct: The class acquires multiple interface types, which is multiple type inheritance.

D - Incorrect: Interface is the construct defining a contract, not the requested relationship concept.

Primary note proof

Study-note section OOP19

> A class can extend a superclass and implement zero or more interfaces. An interface can extend several interfaces.

Study-note section OOP20

> Use an abstract class for shared state, construction and common implementation; use an interface for a capability across unrelated classes and multiple type inheritance.

Study-note section OOP02

Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 32 | lines 12-22

Extending multiple Interface
Multiple inheritance in Java by interface
• If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.
Extending multiple Interface
• A single interface can extend multiple interfaces, below is a simple example.

Object Oriented Programming module | PDF page 31 | lines 9-30

• It provides total abstraction;
• Means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default.
• A class that implement interface must implement all the methods declared in the interface.
• If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
Rules of Interface
• Variables declared in an interface are public, static & final by default.
• Java does not allow you to extend more than one class, However you can implement more than one interfaces in your class.
• class implements interface but an interface extends another interfaces.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 15-1

Computer Science Exit Exam practice collection · Question 105

Question 176 | SM0283

Which of the following concepts in OOP allows a class to inherit attributes and methods from another class?

A. Inheritance

B. Abstraction

C. Encapsulation

D. Polymorphism

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer A: Inheritance

Acquiring another class's methods and fields defines inheritance.

Why each choice is right or wrong

A - Correct: Matches acquisition from another class

B - Incorrect: Essential information exposure

C - Incorrect: Wrapping data/methods together

D - Incorrect: Many forms of behavior, rather than the transfer relation

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Study-note section OOP02

> Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

> What is Inheritance?
> • Inheritance is one of the cornerstones of

object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Question sources

Computer Science Exit Exam · Sene 2016 · Question 27-2

Computer Science Exit Exam practice collection · Question 103

Question 177 | SM0290

According to the introductory module's language-characteristics list, which description contradicts its characterization of Java?

A. Java is Object-Oriented

B. Java is Platform Independent

C. Java is difficult to learn

D. Java is architecture neutral

Hint

Read the cited conceptual rules, state the model, and apply them before comparing the alternatives.

Answer C: Java is difficult to learn

Java is commonly characterized as object oriented, platform independent and simple. Architecture-neutral bytecode is executed by a compatible platform-specific JVM. Difficult to learn conflicts with the module's intended simple characterization, making C the requested mismatch. This is a question about the module's description, not an objective claim that every learner finds Java easy or that every program runs unchanged in every environment.

Why each choice is right or wrong

A - Incorrect: Object orientation is part of the stated introductory characterization.

B - Incorrect: Platform independence is part of the stated characterization, subject to a compatible JVM and environment.

C - Correct: Difficult to learn conflicts with the module's intended simple characterization.

D - Incorrect: Architecture-neutral bytecode supports this characteristic within the specified compatible-runtime model.

Primary note proof

Study-note section OOP-S06

> Java is commonly characterized as object oriented, platform independent and simple. Its architecture-neutral bytecode is executed by a compatible platform-specific JVM. "Simple" is a teaching characterization relative to language design, not a promise that every learner finds every Java program easy.

Study-note section OOP39

> Java source normally compiles to JVM bytecode, and a compatible JVM plus libraries supports portability across systems.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 17-29

> The Java compiler translates the program
> into an intermediate language called Java
> bytecodes.
> Java bytecodes are platform-independent.
> Guaranteed to be Write Once, Run
> Anywhere.
> There are different Java interpreters for
> different platforms.
> Java is: + Object Oriented + Platform
> Independent

+ Simple + Secure
+ Portable +
Multithreaded + Robust

Question sources

Computer Science Exit Exam · Sene 2016 · Question 29-3

Computer Science Exit Exam practice collection · Question 95

Question 178 | SS094

Which statement about Java interfaces is false? In A, “interface instance” means an object made by directly instantiating the interface itself, without an implementation. C means interfaces may include abstract methods.

A. An interface can be instantiated directly without supplying an implementation, and its abstract methods can then be invoked

B. Like a class, an interface is a reference data type

C. Interface includes abstract methods

D. One class can implement multiple interfaces

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer A: An interface can be instantiated directly without supplying an implementation, and its abstract methods can then be invoked

An interface cannot be instantiated directly without an implementation. An interface reference can instead hold an implementing object and invoke its contract methods. Interfaces are reference types, can declare abstract operations, and can be implemented alongside other interfaces by one class. Clarifying A’s construction meaning isolates the false statement without incorrectly rejecting ordinary calls through interface-typed references.

Why each choice is right or wrong

A - Correct: Direct interface instantiation without an implementation is forbidden, so A is false.

B - Incorrect: An interface is a reference type usable for implementation references, so B is true.

C - Incorrect: An interface may declare abstract methods, so C is true.

D - Incorrect: One class can implement multiple interfaces, so D is true.

Primary note proof

Study-note section OOP19

interface Payable{double getPaymentAmount();} declares an abstract public operation. class Invoice implements Payable must provide public double getPaymentAmount(), perhaps

returning getQuantity()*getPricePerItem(). A class can extend a superclass and implement zero or more interfaces. An interface can extend several interfaces.

Study-note section OOP19

Interfaces cannot be instantiated directly, but references may hold implementing objects, anonymous implementations or suitable functional-interface lambdas.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 30 | lines 37-59

Chapter - Five: Java Interface
• An interface in java is a blueprint of a class.
• It has static constants and abstract methods.
• The interface in java is a mechanism to achieve abstraction.
• There can be only abstract methods in the java interface not method body (contain only signature, not body).
• It is used to achieve abstraction and multiple inheritance in Java.
• Java Interface also represents IS-A relationship.
• It cannot be instantiated just like abstract class.
• There are mainly three reasons to use interface.
• It is used to achieve abstraction.
• By interface, we can support the functionality of multiple inheritance.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference

types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Object Oriented Programming module | PDF page 31 | lines 41-68

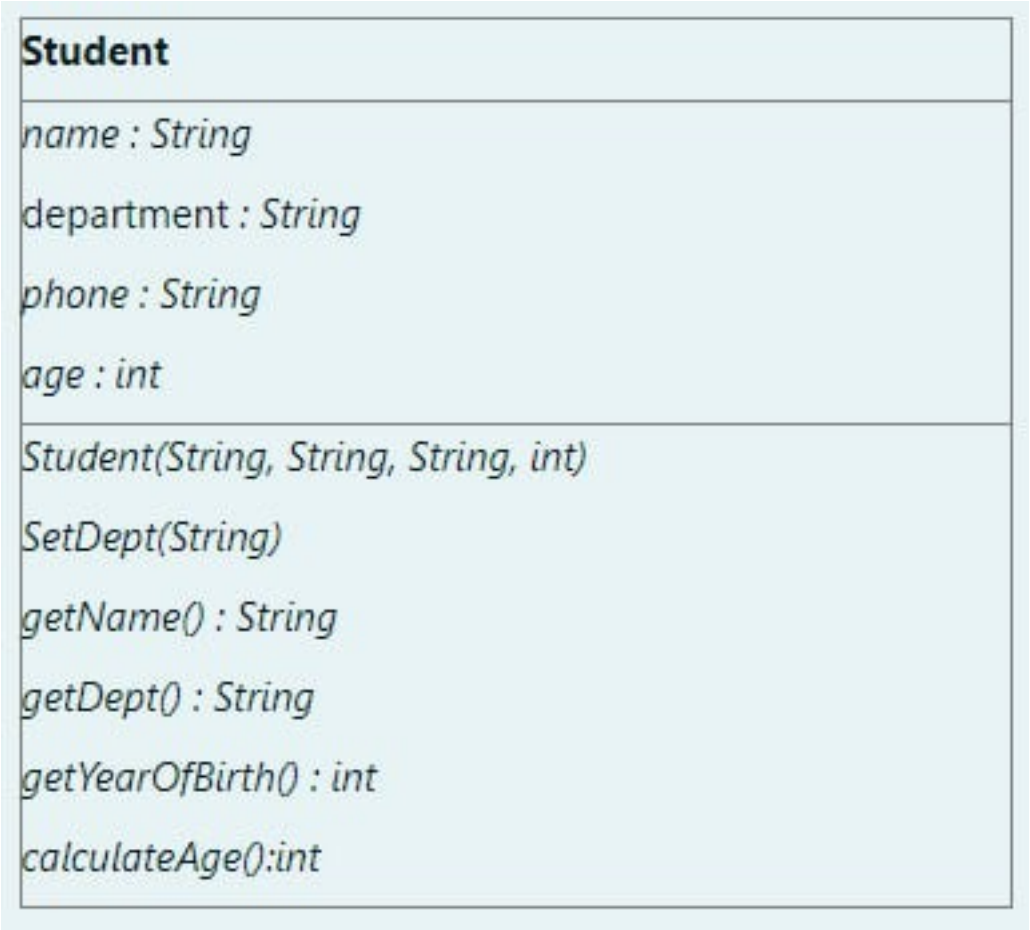
```
Example 2:
interface MyInterface{
/* compiler will treat them
as: public abstract void
method1(); public abstract void
method2(); */
public void method1();
public void method2(); }
class Demo implements
MyInterface{
/* This class must have to
implement both the abstract
methods else you will get
compilation error */
public void method1(){
System.out.println("imple
mentation of method1");
}
public void method2(){
System.out.println("imple
mentation of method2");
}
public static void
main(String arg[]){
MyInterface obj = new
Demo();
obj.method1();}}
Output:implementation of method1
```

Question sources

Computer Science model exam collection · Question 1

Question 179 | SS097

The following UML design shows Student class. Assume that the attributes have declared as private and the methods as public. The get methods
return the value of the attributes and the calculateAge() returns the Student age. Which of the following would not raise an error when you run this
program?



A. Student st1 = new Student ("John","CS","0913222222","25");

B. Student st1 = new Student ("John","SE","0913222222",20);

C. st1 Student = new Student ("John","CS","02085332455", 23);

D. Student st1 = new Student ("John","IT","2000");

Hint

Read the notation and distinguish a class description from a particular instance.

Answer B: Student st1 = new Student ("John","SE","0913222222",20);

The UML specifies Student parameters of types String, String, String and int, in that order. B supplies exactly those four argument types. Distinct String and int primitives/reference types rule out the quoted age in A.

Why each choice is right or wrong

A - Incorrect: Fourth argument is String 25 rather than int

B - Correct: Three String arguments and an int age match the UML

C - Incorrect: Reverses the required type/name declaration order

D - Incorrect: Only three String arguments; missing the required int age

Primary note proof

Study-note section OOP04

A UML class box normally contains name, attributes and operations. Visibility uses + public, - private and # protected. A field can be written name:String and an operation getName():String. A constructor Student(String,String,String,int) requires three strings and an integer, so an age argument "25" has the wrong type.

Study-note section OOP03

Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 12 | lines 24-47

Constructors
• Performs automatic initialization of objects. .:.
• Initializes an object immediately upon creation.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
• The default constructor automatically initializes all instance variables to zero and all object references to NULL.
• The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
• Once you define your own constructor, the default constructor is no longer used.
• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Object Oriented Programming module | PDF page 6 | lines 4-17

Java has two basic data types
1. Primitive Data Types: are 8 in java (byte, short, int, long, float, double, char & Boolean)
2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Object Oriented Programming module | PDF page 8 | lines 43-55

• a class declaration only creates a template; it does not create an actual object.
• To actually create a Dog object, use the second statement.
• Dog myDog; // Creates only the template
• myDog = new Dog(); // create a Dog object called myDog
• After this statement executes, myDog will be an instance of Dog.

Question sources

Computer Science model exam collection · Question 4

Question 180 | SS104

What we use to pass the data between activities in Android?

A. Broadcast receiver

B. PostgreSQL Database

C. Intent

D. Content Provider

Hint

Distinguish an interactive entry point, a requested action and attached data.

Answer C: An Intent, with extras.

OOP35 says an Intent describes an action or target and extras carry values between activities. Its putExtra("id",17) example supplies a concrete transfer. A broadcast receiver handles broadcasts and a content provider shares structured data; neither is the ordinary activity-start-with-parameters mechanism. A PostgreSQL database is not the inter-activity carrier described here.

Why each choice is right or wrong

A - Incorrect: A broadcast receiver handles broadcasts rather than serving as the ordinary activity-to-activity parameter carrier.

B - Incorrect: A database can store data but is not the Intent-and-extras transfer mechanism specified in OOP35.

C - Correct: An Intent selects the action/target and carries values between activities using extras.

D - Incorrect: A content provider shares structured data, rather than being the ordinary parameterized activity-start mechanism.

Primary note proof

Study-note section OOP35

> An Intent describes a requested action or explicit target; extras carry values between activities. For example, putExtra("id",17) attaches an ID for the destination to read. A content provider shares structured data, while a broadcast receiver handles broadcast messages; neither is the ordinary mechanism for starting an activity with parameters.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 17-29

> The Java compiler translates the program
> into an intermediate language called Java
> bytecodes.
> Java bytecodes are platform-independent.
> Guaranteed to be Write Once, Run
> Anywhere.
> There are different Java interpreters for
> different platforms.
> Java is: + Object Oriented + Platform
> Independent
> + Simple + Secure
> + Portable +
> Multithreaded + Robust

Question sources

Computer Science model exam collection · Question 11

Question 181 | SS108

What Is Manifest.xml in android?

A. It has the information about activities in an application

B. It has all the information about an application

C. Itis an executable file of the application

D. It has information about layout in an application

Hint

Separate application declarations, view containers and installation packaging.

Answer A: The manifest declares essential metadata, including application activities.

OOP36 says AndroidManifest.xml declares essential metadata such as components, permissions and intent filters. Activities are components, so A states a supported part of its contents. B overstates the scope by claiming all application information, which the note explicitly rejects. The image's reordered highlighted all-information choice cannot override that rule or the main option labels.

Why each choice is right or wrong

A - Correct: Activities are application components whose declarations form part of the manifest's essential metadata.

B - Incorrect: All information is too broad; the manifest does not contain all source code or layout details.

C - Incorrect: The manifest declares metadata and is explicitly not executable application code.

D - Incorrect: Layout resources describe views; the manifest does not contain the layout details asserted here.

Primary note proof

Study-note section OOP36

> AndroidManifest.xml declares essential app metadata such as components, permissions and intent filters. It is not executable code and does not contain all source or layout details.

Study-note section OOP36

> A manifest describes the package, while layout resources describe views.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 17-29

> The Java compiler translates the program
> into an intermediate language called Java
> bytecodes.
> Java bytecodes are platform-independent.

Guaranteed to be Write Once, Run Anywhere.
There are different Java interpreters for different platforms.
Java is: + Object Oriented + Platform Independent
+ Simple + Secure
+ Portable +
Multithreaded + Robust

Question sources

Computer Science model exam collection · Question 15

Question 182 | SS109

Assume you have an interface called Payable as follows:
public interface Payable {
double getPaymentAmount();
}
If you have another class Invoice that implements this interface and getQuantity() and getPricePerItem() methods that return quantity and price per item, which options correctly implement the Payable contract? Treat each option as an independent class skeleton with accessible getQuantity() and getPricePerItem() methods supplied and compatible with multiplication returning double. An abstract class may supply a concrete implementation. Select all that apply.

A. public abstract class Invoice implements Payable {
public double getPaymentAmount() {
return getQuantity() * getPricePerItem();
}
}

B. public class Invoice implements Payable {
public double getPaymentAmount() {
return getQuantity() * getPricePerItem();
}
}

C. public class Invoice implements Payable {
public getPaymentAmount() {
return getQuantity() * getPricePerItem();
}
}

D. public class Invoice extends Payable {
public double getPaymentAmount() {
return getQuantity() * getPricePerItem();
}
}

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

**Answer A, B: public abstract class Invoice implements Payable {
public double getPaymentAmount() {
return getQuantity() * getPricePerItem();
}
}; public class Invoice implements Payable {
public double getPaymentAmount() {
return getQuantity() * getPricePerItem();
}
}**

Both A and B provide public double getPaymentAmount() with the required quantity-times-unit-price result. Declaring A abstract does not prevent it from implementing the interface method concretely; it only prevents direct construction of Invoice itself. The assumed helper methods make both skeletons complete for this comparison. C omits the return type, while D incorrectly uses extends for the interface relationship. Thus A and B satisfy the contract.

Why each choice is right or wrong

A - Correct: An abstract class may still provide this concrete public double implementation.

B - Correct: The concrete class provides the required public double implementation.

C - Incorrect: The method lacks a return type and is not a valid implementation declaration.

D - Incorrect: A class implements Payable; it cannot extend that interface as a superclass.

Primary note proof

Study-note section OOP19

> interface Payable{double getPaymentAmount();} declares an abstract public operation. class Invoice implements Payable must provide public double getPaymentAmount(), perhaps returning getQuantity()*getPricePerItem(). A class can extend a superclass and implement zero or more interfaces.

Study-note section OOP18

> An abstract class cannot be directly instantiated but can define fields, constructors, abstract methods and concrete methods.

Study-note section OOP11

> A method has modifiers, return type, name, parameter list and body: public int getAge(){return age;}.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 31 | lines 9-30

> • It provides total abstraction;
> • Means all the methods in interface
> are declared with empty body and
> are public and all fields are public,

static and final by default.
• A class that implement interface must implement all the methods declared in the interface.
• If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
Rules of Interface
• Variables declared in an interface are public, static & final by default.
• Java does not allow you to extend more than one class, However you can implement more than one interfaces in your class.
• class implements interface but an interface extends another interfaces.

Object Oriented Programming module | PDF page 23 | lines 57-81

Abstract Classes
• A classes declared with abstract keyword and can‟t be instantiated.
• Used to hide meaningless methods:
• There can be no objects of an abstract class. .:.
• An abstract class cannot be directly instantiated with the new operator.
• You cannot declare abstract constructors, or abstract static methods
• Syntax: abstract class class_name
Example:

```
abstract class A {
abstract void callme();
// concrete methods are still
allowed in abstract classes
void callmetoo() {
System.out.println("This is a
concrete method.");
}}
class B extends A {
void callme() {
System.out.println("B's
implementation of callme.");
}}
```

Question sources

Computer Science model exam collection · Question 16

Computer Science model exam collection

Question 183 | SS117

Which statement is wrong about a Java try-catch construct? Assume any exception considered has a matching catch, and no earlier control transfer prevents reaching it.

A. The execution control is moved to catch block after exception occur in try block

B. When exception occurs, the execution of try block is interrupted

C. The catch block is executed once regardless of exceptions in try block

D. The catch block is used to handle exceptions that occur in try block

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer C: The catch block is executed once regardless of exceptions in try block

If the try block completes normally, its catch does not execute. If an exception occurs, normal execution of the try is interrupted and the matching handler runs. Thus the claim that catch runs once regardless of exceptions is false. Catch is conditional handling, whereas finally normally provides cleanup after the try and any selected handler. The matching-handler assumption makes the other transfer statements precise.

Why each choice is right or wrong

A - Incorrect: With the stated matching-handler assumption, an exception transfers control to that catch.

B - Incorrect: The exception interrupts normal execution and skips the remaining try body.

C - Correct: This is false: normal completion of try does not execute catch.

D - Incorrect: The matching catch handles the exception thrown during try.

Primary note proof

Study-note section OOP30

> try encloses guarded operations; a matching catch handles the thrown type or a subtype. Execution skips the remainder of the try when a throw occurs.

Study-note section OOP31

> finally normally executes after try and any selected catch, including when leaving by return or propagating an exception.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 28 | lines 8-22

```
try {
int num=
reader.nextInt();
System.out.println("The number
is :" + num);
}
catch
(InputMismatchException e){
System.out.println("There is
type mis match :");
}
finally{
System.out.println("This part is
always done");
}} }
```

Object Oriented Programming module | PDF page 29 | lines 11-22

```
Using try and catch...
General form of try/catch:
Try{
//block of code to monitor for
errors
}
Catch(ExceptionType1 exOb ) {
//handler for ExceptionType1
}
Catch(ExceptionType2 exOb ) {
//handler for ExceptionType2
}
```

Object Oriented Programming module | PDF page 27 | lines 20-41

```
• An exception is an event, which occurs
during the execution of a program, that
disrupts the normal flow of the
program's instructions.
• An exception is an indication of a
problem that occurs during a program‟s
execution.
• An exception is a runtime error.
• A Java exception is an instance of a class
derived from Throwable.
• The Throwable class is contained in the
java.lang package, and subclasses of
Throwable are contained in various
packages.
• Errors related to GUI components are
```

included in the java.awt package; numeric exceptions are included in the java.lang package because they are related to the java.lang.Number class.
• You can create your own exception classes by extending Throwable or a subclass of Throwable.

Question sources

Computer Science model exam collection · Question 24

Question 184 | SS118

An accessible abstract Class_A declares public abstract String welcome(); and has an accessible no-argument constructor. Which independent concrete definition of Class_B compiles? Each public class is in its own correctly named source file. These options restore the complete A-D code blocks from the source images.

A.
```
public class Class_B extends Class_A {
  public String whatObjectAmI() { return "I am a class B object"; }
  }
```

B.
```
public class Class_B extends Class_A {
  public int welcome() { int x = 10; return x; }
  }
```

C.
```
public class Class_B extends Class_A {
  public String whatObjectAmI() { return "I am a class D object"; }
  public String welcome() { return "I am a class C object"; }
  public int welcome(int y) { int x = 2 * y; return x; }
  }
```

D.
```
public class Class_B extends Class_A {
  public void welcome(String str) { }
  }
```

Hint

Check whether a class is concrete and whether every inherited abstract contract has an implementation.

Answer C: public class Class_B extends Class_A {
public String whatObjectAmI() { return "I am a class D object"; }
public String welcome() { return "I am a class C object"; }
public int welcome(int y) { int x = 2 * y; return x; }
}

The required inherited method has no parameters and returns String. C supplies that exact public implementation and may additionally overload welcome with an int parameter. A has no welcome implementation, B returns an incompatible int, and D supplies only a different-parameter overload. Those concrete subclasses leave the abstract contract unsatisfied or violate return compatibility.

Why each choice is right or wrong

A - Incorrect: whatObjectAmI() does not implement the required welcome() contract.

B - Incorrect: int is incompatible with the required String return for the same signature.

C - Correct: The exact public String welcome() implementation is present; welcome(int) is an allowed overload.

D - Incorrect: welcome(String) is an overload and does not implement no-argument welcome().

Primary note proof

Study-note section OOP18

> A concrete subclass must provide or inherit implementations of all required abstract methods; otherwise it must itself be abstract. If parent declares abstract String welcome(), a String welcome() implementation is required; int welcome() is incompatible, and welcome(int) merely adds an overload. A valid class can contain both the required implementation and an unrelated overload.

Study-note section OOP12

> Overloaded methods share a name but have different parameter types, counts or type order. Return type alone is insufficient in Java and C++.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 23 | lines 57-81

> Abstract Classes
> • A classes declared with abstract
> keyword and can‟t be instantiated.
> • Used to hide meaningless methods:
> • There can be no objects of an abstract
> class. .:.
> • An abstract class cannot be directly
> instantiated with the new operator.
> • You cannot declare abstract
> constructors, or abstract static methods

• Syntax: abstract class class_name
Example:
abstract class A {
abstract void callme();
// concrete methods are still
allowed in abstract classes
void callmetoo() {
System.out.println("This is a
concrete method.");
}}
class B extends A {
void callme() {
System.out.println("B's
implementation of callme.");
}}

Object Oriented Programming module | PDF page 11 | lines 55-67

Method Overriding
In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, this is called method Overriding.
• the method in the subclass is said to override the method in the superclass.
• The Method in the super class is known as an overridden method which is called within a subclass.
• The version of the method defined by the superclass will be hidden.

Object Oriented Programming module | PDF page 11 | lines 7-18

Method Overloading
• Is a process of having the same name for different methods as long as their parameter list or data types in the parameter list is different?
• The return type alone is insufficient to distinguish two versions of a method.
• Parameters are key identifiers of OM.
• Is one of the ways that Java supports polymorphism

Question sources

Computer Science model exam collection · Question 25

Computer Science model exam collection

Question 185 | SS119

For traditional Android View-based layout containers, which class is their common container superclass?

A. android.view.Layout

B. android.view.RelativeLayout

C. android.view.ViewGroup

D. android.view.Widget

Hint

Separate application declarations, view containers and installation packaging.

Answer C: android.view.ViewGroup, for traditional view-based layout containers.

Traditional Android View-based layout containers, including LinearLayout and RelativeLayout, derive from android.view.ViewGroup. ViewGroup contains child views, matching the container role. RelativeLayout is one particular container rather than the common superclass, and the other supplied names do not identify that superclass. The View-system scope avoids extending this claim to every modern Android layout abstraction.

Why each choice is right or wrong

A - Incorrect: This is not the named View-system container superclass.

B - Incorrect: RelativeLayout is a particular layout container, not the common superclass of the layout containers.

C - Correct: android.view.ViewGroup is the child-view container superclass for this traditional View-system scope.

D - Incorrect: This does not name the View-system container superclass.

Primary note proof

Study-note section OOP36

> Traditional view-based layout containers such as LinearLayout and RelativeLayout derive from android.view.ViewGroup, which contains child views.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 17-29

The Java compiler translates the program into an intermediate language called Java bytecodes.
Java bytecodes are platform-independent.
Guaranteed to be Write Once, Run Anywhere.
There are different Java interpreters for different platforms.
Java is: + Object Oriented + Platform Independent
+ Simple + Secure
+ Portable +
Multithreaded + Robust

Question sources

Computer Science model exam collection · Question 26

Question 186 | SS121

Which statements about a Java abstract class are true? Select all that apply.

A. Cannot be instantiated directly

B. Must contain at least one abstract method

C. Can contain both abstract and non-abstract methods

D. Cannot be inherited

Hint

Check whether a class is concrete and whether every inherited abstract contract has an implementation.

Answer A, C: Cannot be instantiated directly; Can contain both abstract and non-abstract methods

An abstract class cannot be constructed directly, but it can contain both abstract and concrete methods. It can also be abstract without declaring any abstract method, so B is false. Abstract classes may be inherited; a concrete subclass must fulfill outstanding abstract method contracts. Cleaning the source marking noise and replacing the vague encapsulation slogan leaves two precise true properties at A and C.

Why each choice is right or wrong

A - Correct: Direct instantiation of an abstract class is prohibited.

B - Incorrect: An abstract class need not declare any abstract method.

C - Correct: Abstract and concrete methods may coexist in an abstract class.

D - Incorrect: Abstract classes can be inherited; concrete subclasses must fulfill their contracts.

Primary note proof

Study-note section OOP18

An abstract class cannot be directly instantiated but can define fields, constructors, abstract methods and concrete methods. It need not contain an abstract method. An abstract method has no body; abstract and final conflict because an operation cannot require overriding while forbidding it.

Study-note section OOP18

A concrete subclass must provide or inherit implementations of all required abstract methods; otherwise it must itself be abstract.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 23 | lines 57-81

Abstract Classes
• A classes declared with abstract
keyword and can‟t be instantiated.
• Used to hide meaningless methods:
• There can be no objects of an abstract
class. .:.
• An abstract class cannot be directly
instantiated with the new operator.
• You cannot declare abstract
constructors, or abstract static methods
• Syntax: abstract class class_name
Example:
abstract class A {
abstract void callme();
// concrete methods are still
allowed in abstract classes
void callmetoo() {
System.out.println("This is a
concrete method.");
}}
class B extends A {
void callme() {
System.out.println("B's
implementation of callme.");
}}

Question sources

Computer Science model exam collection · Question 28

Question 187 | SS129

What is an activity in android?

A. A single screen in an application with supporting java code

B. android class used to configure the android application

C. Itis an intent

D. android package file holding all the packages used

Hint

Distinguish an interactive entry point, a requested action and attached data.

Answer A: An Activity is a user-interaction entry component, often a screen.

An Android Activity is a component providing an entry point for user interaction, often corresponding to a screen. A is the introductory screen-and-code description that best matches this definition, though a one-to-one screen mapping is not universal. Configuration, Intents and package archives are separate concepts.

Why each choice is right or wrong

A - Correct: The screen with supporting code is the introductory description closest to the note's interaction-entry component.

B - Incorrect: Configuring the application describes a different responsibility than providing the Activity's user-interaction entry point.

C - Incorrect: An Intent describes a requested action or target; it is not itself an Activity.

D - Incorrect: An Android package archive is installation/runtime packaging, rather than a user-interaction component.

Primary note proof

Study-note section OOP35

> An Android Activity is a component providing an entry point for user interaction, often corresponding to a screen. An Intent describes a requested action or explicit target; extras carry values between activities. For example, putExtra("id",17) attaches an ID for the destination to read. A content provider shares structured data, while a broadcast receiver handles broadcast messages; neither is the ordinary mechanism for starting an activity with parameters.

Study-note section OOP36

> An .apk is an Android package archive used for installation and runtime contents.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 17-29

> The Java compiler translates the program
> into an intermediate language called Java

bytecodes.
Java bytecodes are platform-independent.
Guaranteed to be Write Once, Run
Anywhere.
There are different Java interpreters for
different platforms.
Java is: + Object Oriented + Platform
Independent
+ Simple + Secure
+ Portable +
Multithreaded + Robust

Question sources

Computer Science model exam collection · Question 36

Question 188 | SS140

Assume that there is a Pet family from which Pets like Dog and Cat share behavior from.Every Pet family has a name. However, the pet families have
some specific behaviors that are exhibited by only particular Pets. For instance, only Dogs can fetch something while both cats and dogs can speak.
Which of the following object oriented designprinciple would be suitable for this case?

A. Polymorphism

B. Encapsulation

C. Inheritance

D. _ Information hiding

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer C: Inheritance

A shared Pet superclass can hold name and common speak behavior; Dog and Cat specialize it, with fetch only in Dog. This directly uses inheritance. Overriding speak could also yield polymorphism but is not required by the stem.

Why each choice is right or wrong

A - Incorrect: Would apply if speak were dispatched to different overrides, which is not stated

B - Incorrect: Groups data/methods but does not establish shared family specialization

C - Correct: Models shared parent features plus subclass-specific behavior

D - Incorrect: Hides details rather than defining the family hierarchy

Primary note proof

Study-note section OOP13

Use inheritance when a subtype can satisfy the parent's contract: Dog is an Animal. For shared Pet.name and speak(), put common state/contract in Pet, specialized fetch() in Dog, and appropriate speak() implementations in Dog and Cat. Composition represents has-a, such as Car containing Engine. Reuse alone does not justify an invalid is-a relationship.

Study-note section OOP02

Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

What is Inheritance?
• Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{

.....}
class Sub extends Super{ }

Object Oriented Programming module | PDF page 16 | lines 51-70

Types of inheritance
1. Simple/single inheritance: is damn easy to understand. When a class extends another one class only.
2. Multi-Level inheritance: One can inherit from a derived class, thereby making this derived class the base class for the new class.
3. Hierarchical inheritance: One class is inherited by many sub classes.
• Java does not support the inheritance of multiple super classes into a single subclass. (Multiple & Hybrid inheritances)
• No class can be a superclass of itself
• Superclass can be used by itself.
• The superclass has no knowledge of what a subclass adds to it.
• Constructors are not members, so they are not inherited by subclasses.

Object Oriented Programming module | PDF page 3 | lines 58-64

2. Polymorphism:
• The word polymorphism means the ability to take more than one form.
• In terms of the OOP, this means that a particular operation may behave differently for different sub-classes of the same class.

Question sources

Computer Science model exam collection · Question 47

Question 189 | SS165

Which method has the same name as that of its class?

A. constructor

B. class

C. delete

D. finalize

Hint

Check the class name, return-type absence and the constructors actually declared.

Answer A: constructor

The constructor naming rule requires the same name as its class. The term method is used informally in the question; constructors have no return type.

Why each choice is right or wrong

A - Correct: Matches the constructor naming rule

B - Incorrect: class is a declaration keyword, not this routine category

C - Incorrect: delete is not the constructor category

D - Incorrect: finalize has its own name rather than the class name

Primary note proof

Study-note section OOP09

> A constructor has the class's exact name and no return type, not even void.

Study-note section OOP-S05

> In Java's class-declaration terminology, constructors are not members and are not inherited.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 12 | lines 24-47

> Constructors
> • Performs automatic initialization of objects. .:.
> • Initializes an object immediately upon creation.
> • Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
> • The default constructor automatically initializes all instance variables to zero and all object references to NULL.
> • The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
> • Once you define your own constructor, the default constructor is no longer used.

• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Question sources

Computer Science model exam collection · Question 72

Question 190 | SS186

APK stands for:

A. Android Phone Kit

B. Android Page Kit

C. Android Platform Kit

D. Android Package Kit

Hint

Separate application declarations, view containers and installation packaging.

Answer D: Android Package Kit is the common exam expansion.

Android Package Kit is a common exam expansion of APK; the artifact is an Android package archive. Thus D is appropriate under this expansion convention. None of the Phone, Page or Platform alternatives describes this package archive.

Why each choice is right or wrong

A - Incorrect: Phone Kit does not match the common expansion or the package-archive meaning given in OOP36.

B - Incorrect: Page Kit does not describe the installation package archive identified here.

C - Incorrect: Platform Kit is not the bank expansion explicitly documented in OOP36.

D - Correct: Package Kit is the common exam expansion expressly stated, with Android package as the official artifact terminology.

Primary note proof

Study-note section OOP36

An .apk is an Android package archive used for installation and runtime contents. “Android Package Kit” is a common exam expansion; Android’s documentation calls it an Android package.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 2 | lines 17-29

The Java compiler translates the program into an intermediate language called Java bytecodes.
Java bytecodes are platform-independent.
Guaranteed to be Write Once, Run Anywhere.
There are different Java interpreters for different platforms.
Java is: + Object Oriented + Platform Independent
+ Simple + Secure
+ Portable +
Multithreaded + Robust

Question sources

Computer Science model exam collection · Question 93

Question 191 | AM0304

Which option specifically describes encapsulation’s bundling of related state and operations, rather than focusing on hiding implementation decisions behind an interface?

A. Combining data and methods into a single entity.

B. Hiding implementation details and providing a public interface.

C. Creating multiple instances of a class.

D. Inheriting properties and behavior from a base class.

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer A: Combining data and methods into a single entity.

Encapsulation groups related state and operations and controls access to them. A directly describes the grouping or bundling aspect requested here. B describes information hiding and can accompany encapsulation, but the stem deliberately asks for the grouping emphasis. Creating instances is object construction, while inheriting base properties is inheritance. This specific emphasis makes A the unique answer without denying the overlap of the broader principles.

Why each choice is right or wrong

A - Correct: Combining data and methods directly describes the requested bundling aspect.

B - Incorrect: This emphasizes information hiding rather than the specific grouping mechanism asked for.

C - Incorrect: Creating instances is object construction rather than encapsulation’s grouping.

D - Incorrect: Reusing base-class properties describes inheritance.

Primary note proof

Study-note section OOP02

Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller. These ideas overlap: a private balance plus deposit(amount) can demonstrate all three, depending on what the question emphasizes.

Study-note section OOP13

Use inheritance when a subtype can satisfy the parent's contract: Dog is an Animal.

Study-note section OOP03

Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 35-45

4. Encapsulation:
• means the wrapping up of data and methods into a single unit (a class).
• It also means that the data inside a class is hidden from everything outside the class.
• The data can only be accessed by invoking the methods of the class.
• It is placing the data and the functions that work on that data in the same place.

Object Oriented Programming module | PDF page 3 | lines 27-33

0. Abstraction:
• provides only essential information to the outside world.
• Or hiding their background details, i.e., to represent the needed information in program without presenting the details.

Question sources

Computer Science MCQ practice collection · Question 304

Question 192 | AM0305

What is inheritance in Java?

A. The process of creating objects from classes.

B. The ability of a class to inherit properties and behavior from another class.

C. Creating multiple instances of a class.

D. The ability to access variables and methods of a different class.

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer B: The ability of a class to inherit properties and behavior from another class.

Inheritance means acquiring another class's fields and methods, matching the properties-and-behavior description.

Why each choice is right or wrong

A - Incorrect: Instantiation rather than inheritance

B - Correct: Matches acquisition from a parent class

C - Incorrect: Instance creation rather than inheritance

D - Incorrect: Too broad: arbitrary access to another class is not inheritance

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Study-note section OOP03

> Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance. Member access uses s.name or s.study(), subject to visibility.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

> What is Inheritance?
> • Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
> • Inheritance is the process where one class acquires the properties (methods and fields) of another.
> • A class that is inherited is called a

superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Question sources

Computer Science MCQ practice collection · Question 305

Question 193 | AM0306

What is the concept of polymorphism in Java?

A. The ability to create multiple objects from a single class.

B. The ability to hide implementation details.

C. The ability to override methods in a subclass.

D. The ability of an object to take many forms and be treated as an instance of its superclass.

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer D: The ability of an object to take many forms and be treated as an instance of its superclass.

Polymorphism means many forms and can be illustrated by a parent reference to a child object. Overriding is one mechanism, not the full concept.

Why each choice is right or wrong

A - Incorrect: Creating several same-class objects does not imply many type forms

B - Incorrect: Hiding details describes abstraction

C - Incorrect: Overriding supports runtime polymorphism but is a narrower mechanism

D - Correct: Matches both many forms and common-superclass treatment

Primary note proof

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Study-note section OOP15

> For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile. If fetch() exists only in Dog, a.fetch() fails compilation. For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Study-note section OOP02

> Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 25 | lines 17-37

> • Polymorphism is the ability of an object to take on many forms.
> • The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
> • Any Java object that can pass more than one IS-A test is considered to be polymorphic.
> • In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.
> • Polymorphism is derived from 2 Greek words: poly and morphs.
> • The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
> • polymorphism is the capability of an action or method to do different things based on the object that it is acting upon.

Object Oriented Programming module | PDF page 11 | lines 55-67

Method Overriding
In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, this is called method Overriding.
• the method in the subclass is said to override the method in the superclass.
• The Method in the super class is known as an overridden method which is called within a subclass.
• The version of the method defined by the superclass will be hidden.

Object Oriented Programming module | PDF page 3 | lines 27-33

0. Abstraction:
• provides only essential information to the outside world.
• Or hiding their background details, i.e., to represent the needed information in program without presenting the details.

Question sources

Computer Science MCQ practice collection · Question 306

Question 194 | AM0307

What is the purpose of the super keyword in Java?

A. To access the superclass of a derived class.

B. To instantiate a new object.

C. To access the current object.

D. To define a method within a class.

Hint

Distinguish the current instance, parent context, hidden fields and static dispatch.

Answer A: To access the superclass of a derived class.

super selects the superclass context for member access and can request a superclass implementation. A hidden parent field can be selected with super.name, and super.sound() directly calls the parent version. It does not allocate a separate parent object or name the current instance; this is the current-object reference. Method definitions use declarations, so A is the valid purpose among the primary choices.

Why each choice is right or wrong

A - Correct: This describes selecting the superclass context and its members.

B - Incorrect: super does not allocate a separate object.

C - Incorrect: this denotes the current object; super selects parent context.

D - Incorrect: A method declaration defines a method; super does not do that.

Primary note proof

Study-note section OOP16

> Java this denotes the current object; this.name=name distinguishes a field from a parameter. super.name selects a hidden superclass field; super.sound() directly requests the parent implementation. It does not create a second parent object. An unqualified instance call may dispatch dynamically; an explicit super call is different.

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 17 | lines 51-74

> The super keyword
> • Since encapsulation is a primary attribute of OOP, Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.
> • Super has two general forms.
> 1. Used to access parent class instance members.
> 2. Used to access parent class constructor.
> 1. Used to access parent class instance members.
> • super acts somewhat like this, except that it always refers to the superclass of the subclass in which it is used.
> • Super has the form: super.member Here, member can be either a method or an instance variable.
> • Super is most applicable to resolve member hiding.
> • super( ) must always be the first statement executed inside a subclass‟ constructor.

Variant 1 | Alternative version 1

Inside an instance method declared in the current subclass, which descriptions correctly identify the parent-class context selected by Java super? Select all that apply.

A. It refers to the current object instance.

B. It refers to the superclass of the current class.

C. It refers to the class that contains the current method.

D. It refers to the class from which the current class is derived.

Answer B, D: Both B and D describe the superclass context selected by super.

The superclass of the current class and the class from which the current class derives name the same parent relationship, so B and D both apply. super selects that parent context for member access or explicit parent implementation calls. A instead describes this, while C identifies the current subclass containing the declared method. The variant therefore needs multiple selection even though the primary choices have one correct answer.

Why each choice is right or wrong

A - Incorrect: The current object instance is denoted by this, not the parent-context selector super.

B - Correct: The superclass is the parent context selected by super.

C - Incorrect: The method is declared in the current subclass, whereas super selects its parent context.

D - Correct: The class from which the current class derives is its superclass, equivalent to B.

Primary note proof

Study-note section OOP16

> Java this denotes the current object; this.name=name distinguishes a field from a parameter. super.name selects a hidden superclass field; super.sound() directly requests the parent implementation. It does not create a second parent object. An unqualified instance call may dispatch dynamically; an explicit super call is different.

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Question sources

Computer Science MCQ practice collection · Question 307

Computer Science MCQ practice collection · Question 332

Question 195 | AM0308

What is the difference between an abstract class and an interface in Java?

A. An abstract class can have concrete methods, while an interface cannot.

B. An interface can have static methods, while an abstract class cannot.

C. An abstract class can be instantiated, while an interface cannot.

D. An interface can extend multiple interfaces, while an abstract class cannot.

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer D: Interfaces can extend multiple interfaces; an abstract class instead implements interfaces and extends one class.

OOP19 permits an interface to extend several interfaces and a class to extend one superclass while implementing multiple interfaces. Thus D is correct when extend retains its Java declaration meaning. It must not be paraphrased as abstract classes cannot implement multiple interfaces. A is outdated for modern interfaces, B wrongly excludes class static methods, and C wrongly permits abstract instantiation.

Why each choice is right or wrong

A - Incorrect: Modern interfaces can contain default/static bodies, so concrete behavior is not exclusive to abstract classes.

B - Incorrect: Abstract classes can have static methods; their abstract modifier does not prohibit class-level methods.

C - Incorrect: An abstract class cannot be directly instantiated, so the first half of this distinction is false.

D - Correct: An interface extends multiple interfaces; a class implements them instead and has one direct superclass.

Primary note proof

Study-note section OOP19

> A class can extend a superclass and implement zero or more interfaces. An interface can extend several interfaces.

Study-note section OOP19

> A method declaration without private, default or static is implicitly public abstract. Modern interfaces may also have default/static method bodies, and Java 9+ private helper methods.

Study-note section OOP18

> An abstract class cannot be directly instantiated but can define fields, constructors, abstract methods and concrete methods.

Study-note section OOP07

static gives one field per loaded class, shared by its instances. If count starts at zero and every constructor increments it, three creations leave count=3. Prefer ClassName.count and ClassName.utility().

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 23 | lines 57-81

Abstract Classes
• A classes declared with abstract keyword and can‟t be instantiated.
• Used to hide meaningless methods:
• There can be no objects of an abstract class. .:.
• An abstract class cannot be directly instantiated with the new operator.
• You cannot declare abstract constructors, or abstract static methods
• Syntax: abstract class class_name
Example:
abstract class A {
abstract void callme();
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
}}
class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
}}

Object Oriented Programming module | PDF page 30 | lines 37-59

Chapter - Five: Java Interface
• An interface in java is a blueprint of a class.
• It has static constants and abstract methods.
• The interface in java is a mechanism to achieve abstraction.
• There can be only abstract methods in the java interface not method body (contain only signature, not body).
• It is used to achieve abstraction and multiple inheritance in Java.

• Java Interface also represents IS-A relationship.
• It cannot be instantiated just like abstract class.
• There are mainly three reasons to use interface.
• It is used to achieve abstraction.
• By interface, we can support the functionality of multiple inheritance.

Object Oriented Programming module | PDF page 32 | lines 12-22

Extending multiple Interface
Multiple inheritance in Java by interface
• If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.
Extending multiple Interface
• A single interface can extend multiple interfaces, below is a simple example.

Question sources

Computer Science MCQ practice collection · Question 308

Question 196 | AM0309

Which access modifier provides the widest accessibility in Java?

A. private

B. protected

C. public

D. default (no modifier)

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer C: public

World-visible public access is wider than class-only, package-only or package-plus-subclasses access.

Why each choice is right or wrong

A - Incorrect: Class-only

B - Incorrect: Package and subclasses

C - Correct: Visible to the world

D - Incorrect: Package-only

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

> Access Specifiers
> Access level modifiers determine whether
> other classes can use a particular field or
> invoke a particular method.
> Their are four types access levels:
> 1. No Modifier(Default): Visible to
> the package(source file), and it is
> called package-private
> 2. Private: Visible to the class only.
> 3. Public: Visible to the world.
> 4. Protected: Visible to the package
> and all subclasses.

Question sources

Computer Science MCQ practice collection · Question 309

Question 197 | AM0310

What is the purpose of the this keyword in Java?

A. To refer to the superclass of a class.

B. To instantiate a new object.

C. To access the current object's variables and methods.

D. To define a method within a class.

Hint

Distinguish the current instance, parent context, hidden fields and static dispatch.

Answer C: Access the current object's variables and methods.

OOP16 says this denotes the current object and demonstrates this.name=name to distinguish a field from a parameter. That makes C the requested purpose. super selects parent context, new creates an object, and method declarations define methods. this neither creates a new instance nor denotes the superclass or the class itself.

Why each choice is right or wrong

A - Incorrect: Superclass context is selected using super, whereas this denotes the current object.

B - Incorrect: Object instantiation uses new and constructor invocation, not the this current-object reference.

C - Correct: this refers to the current object, enabling its instance fields and methods to be selected.

D - Incorrect: A method is defined by a declaration and body; this is an expression referring to the current object.

Primary note proof

Study-note section OOP16

> Java this denotes the current object; this.name=name distinguishes a field from a parameter. super.name selects a hidden superclass field; super.sound() directly requests the parent implementation. It does not create a second parent object.

Study-note section OOP03

> Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 17 | lines 51-74

> The super keyword
> • Since encapsulation is a primary attribute of OOP, Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.
> • Super has two general forms.
> 1. Used to access parent class instance members.
> 2. Used to access parent class constructor.
> 1. Used to access parent class instance members.
> • super acts somewhat like this, except that it always refers to the superclass of

the subclass in which it is used.
• Super has the form: super.member Here, member can be either a method or an instance variable.
• Super is most applicable to resolve member hiding.
• super( ) must always be the first statement executed inside a subclass‟ constructor.

Object Oriented Programming module | PDF page 18 | lines 14-29

```
// Using super to overcome name
hiding.
class A {
int i; }
class B extends A {
int i; // this i hides the i in
A
B(int a, int b) {
super.i = a; // i in A
i = b; // i in B }
void show() {
System.out.println("i in
superclass: " + super.i);
System.out.println("i in
subclass: " + i);
}}
```

Question sources

Computer Science MCQ practice collection · Question 310

Question 198 | AM0311

Which keyword is used to explicitly indicate that a method does not return any value in Java?

A. void

B. null

C. final

D. static

Hint

Separate parameters from arguments and operations that read state from operations that change it.

Answer A: void

The general method pattern returns a value only if type is not void, and test() is shown as void without a value return.

Why each choice is right or wrong

A - Correct: Explicit no-value-return type

B - Incorrect: null is not the no-value method return type shown

C - Incorrect: final prohibits modification/overriding, not returning a value

D - Incorrect: static belongs to the class, not a no-return indicator

Primary note proof

Study-note section OOP11

> A method has modifiers, return type, name, parameter list and body: public int getAge(){return age;}. A void method returns no value.

Study-note section OOP07

> final prevents reassignment of a variable, overriding of a method or extension of a class.

Study-note section OOP07

> static gives one field per loaded class, shared by its instances.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 10 | lines 62-72

> Methods
> • Are members of classes and do
> operations.
> • Allows different classes to
> communicate each other.
> • E.g. type name(parameter-
> list)
> { // body of method
> //....
> return value; // if
> type is not void

Object Oriented Programming module | PDF page 11 | lines 24-41

> class OverloadDemo {
> void test() {
> System.out.println("No
> parameters"); }
> void test(int a) { // Overload
> test for one integer parameter.

```
System.out.println("a: " + a); }
void test(int a, int b) { //
Overload test for two integer
parameters.
System.out.println("a and b: " +
a + " " + b); }
double test(double a) { //
overload test for a double
parameter
System.out.println("double a: "
+ a);
return a*a; } }
```

Question sources

Computer Science MCQ practice collection · Question 311

Question 199 | AM0312

What is method overloading in Java?

A. Defining multiple methods with the same name but different parameters in the same class.

B. Defining a method within another method.

C. Invoking a method from within itself.

D. Creating multiple instances of a class.

Hint

Compare names and parameter signatures before considering return types or access modifiers.

Answer A: Defining multiple methods with the same name but different parameters in the same class.

The definition retains one method name while differentiating parameter lists. Return type alone is insufficient, so signature differences must be in parameters.

Why each choice is right or wrong

A - Correct: Matches same name and different parameters

B - Incorrect: Nested method definition is not overloading

C - Incorrect: Self-invocation is recursion

D - Incorrect: Multiple objects are instantiation, not overloaded methods

Primary note proof

Study-note section OOP12

Overloaded methods share a name but have different parameter types, counts or type order. Return type alone is insufficient in Java and C++. Overloads can also arise through inherited methods; “same class only” is a teaching simplification. f(int), f(double) and f(int,int) are distinct, but changing a parameter name does not create an overload.

Study-note section OOP03

Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 11 | lines 7-18

Method Overloading
• Is a process of having the same name for different methods as long as their parameter list or data types in the parameter list is different?
• The return type alone is insufficient to distinguish two versions of a method.
• Parameters are key identifiers of OM.
• Is one of the ways that Java supports polymorphism

Variant 1 | Alternative version 1

What is method overloading in Java?

A. Creating multiple methods with the same name but different return types.

B. Creating multiple methods with the same name but different parameter lists.

C. Creating multiple methods with the same name and parameter list but different return types.

D. Creating multiple methods with the same name and parameter list but different access modifiers.

Answer B: Creating multiple methods with the same name but different parameter lists.

Apply the same study-note rule to this alternative source set; the choice contrasts below show the derivation.

Why each choice is right or wrong

A - Incorrect: Same-name/different-return methods are not necessarily overloads; return type alone is explicitly insufficient

B - Correct: Same name with different parameter lists is the exact overloading definition

C - Incorrect: Same parameter list prevents return-type-only overloading

D - Incorrect: Access-modifier differences do not supply the required parameter difference

Primary note proof

Study-note section OOP12

> Overloaded methods share a name but have different parameter types, counts or type order. Return type alone is insufficient in Java and C++.

Study-note section OOP12

> A return type is not part of an ordinary Java method signature.

Related reading (context, not primary note proof)

Study-note section OOP03

Question sources

Computer Science MCQ practice collection · Question 312

Computer Science MCQ practice collection · Question 330

Question 200 | AM0313

What is the purpose of the new keyword in Java?

A. To instantiate a new object.

B. To invoke a method.

C. To define a variable.

D. To define a class.

Hint

Separate a reference declaration, object allocation, reference copying and changes to the shared object.

Answer A: To instantiate a new object.

A reference declaration is distinct from new Dog(), which creates the actual instance and triggers initialization.

Why each choice is right or wrong

A - Correct: Matches object creation

B - Incorrect: An ordinary method call does not by itself state the new allocation

C - Incorrect: A variable declaration can exist without an object

D - Incorrect: class declares the blueprint, not its instance

Primary note proof

Study-note section OOP03

> Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance.

Study-note section OOP09

A constructor has the class's exact name and no return type, not even void. It initializes the new object and can be overloaded or private.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 8 | lines 43-55

• a class declaration only creates a template; it does not create an actual object.
• To actually create a Dog object, use the second statement.
• Dog myDog; // Creates only the template
• myDog = new Dog(); // create a Dog object called myDog
• After this statement executes, myDog will be an instance of Dog.

Object Oriented Programming module | PDF page 12 | lines 24-47

Constructors
• Performs automatic initialization of objects. .:.
• Initializes an object immediately upon creation.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
• The default constructor automatically initializes all instance variables to zero and all object references to NULL.
• The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
• Once you define your own constructor, the default constructor is no longer used.
• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Question sources

Computer Science MCQ practice collection · Question 313

Question 201 | AM0314

What is the output of the following Java code snippet?

```
class MyClass {
public static void main(String[] args) {
int x = 5;
System.out.println(x++);
System.out.println(++x);
}
}
```

A. 5, 6

B. 5, 7

C. 6, 6

D. 6, 7

Hint

Track the old and updated values separately and distinguish a character operand from a string.

Answer B: 5, then 7, on separate lines.

OOP24 gives this exact sequence: postfix x++ supplies the old value 5 and then stores 6; prefix ++x then stores and supplies 7. The two println calls print those values on separate lines. The commas in the choices describe the sequence rather than literal commas emitted by the program.

Why each choice is right or wrong

A - Incorrect: The second prefix increment advances the stored 6 to 7 before printing, so the second value is not 6.

B - Correct: Postfix prints the old 5, then prefix increments the stored 6 and prints 7.

C - Incorrect: The first postfix expression supplies 5 rather than the incremented 6.

D - Incorrect: The second value is 7, but the first remains 5 because its increment is postfix.

Primary note proof

Study-note section OOP24

> Starting x=5, printing x++ then ++x gives 5 then 7.

Study-note section OOP-S02

> A print call emits its value without inserting a separator or ending the line. A println call emits its value followed by a line terminator. Spaces must come from the supplied text or explicit output operations; adjacent calls do not automatically separate their values.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 4 | lines 21-34

✓ Operators: Are symbols that take one or more arguments (operands) and operates on them to produce a result. An operator performs a function on one (Unary), two (Binary), or three operands (Ternary). Java has one ternary operator, ?:, which is a short-hand if-else statement. E.g. c=x?w:s; Operators can be Arithmetic, Logical, assignment, Relational, conditional, Increment,

Question sources

Computer Science MCQ practice collection · Question 314

Question 202 | AM0315

Which Java keyword directly exits the enclosing loop prematurely while allowing execution to continue after that loop in the same method?

A. break

B. continue

C. return

D. exit

Hint

Trace condition tests, control transfers and printing as separate steps.

Answer A: break

break exits the enclosing loop and transfers control to the statement after it. continue skips the remaining body of the current iteration and may begin another iteration. return leaves the entire method, so it does not satisfy the stated requirement to continue after the loop in that same method. exit is not a Java loop-control keyword. The narrower control-flow target makes break unique.

Why each choice is right or wrong

A - Correct: break exits the loop and resumes after it within the method.

B - Incorrect: continue proceeds toward another iteration rather than exiting the loop.

C - Incorrect: return exits the entire method, contrary to the required continuation.

D - Incorrect: exit is not a Java loop-control keyword.

Primary note proof

Study-note section OOP25

> break exits the loop; continue skips its remaining body and then runs a for update.

Study-note section OOP-S03

> An unlabeled break leaves the innermost enclosing loop or switch. continue moves to the next iteration step, while return leaves the current method and may supply a return value. These transfers have different destinations; return is not merely a loop exit.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 7 | lines 26-41

> Iteration: The for(), while() and
> do...while() statements
> 4. Jumping Statements: to jump out
> of loops and to control other areas of
> program java uses break and
> continue statements.
> The break statement: used to exit a
> loop
> The Continue Statement: used to
> force program to go back to the top
> of a loop
> Labels: Java does not include a goto
> statement.
> Instead of goto, Java allows you to
> combine break and continue with a
> label.

Question sources

Computer Science MCQ practice collection · Question 315

Question 203 | AM0316

What is the purpose of the try-catch block in Java?

A. To handle exceptions and prevent program crashes.

B. To define a custom class.

C. To declare variables with restricted access.

D. To instantiate objects.

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer A: To handle exceptions and prevent program crashes.

An exception disrupts execution, and try/catch monitors code and supplies a handler, as shown for an input mismatch. It can handle matched exceptions, not guarantee prevention of every crash.

Why each choice is right or wrong

A - Correct: Matches exception handling, with no universal recovery guarantee

B - Incorrect: Class definition is a different construct

C - Incorrect: Visibility modifiers govern access

D - Incorrect: new and constructors instantiate objects

Primary note proof

Study-note section OOP30

> try encloses guarded operations; a matching catch handles the thrown type or a subtype. Execution skips the remainder of the try when a throw occurs.

Study-note section OOP03

> Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance.

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 27 | lines 20-41

> • An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
> • An exception is an indication of a problem that occurs during a program‟s execution.
> • An exception is a runtime error.
> • A Java exception is an instance of a class derived from Throwable.

• The Throwable class is contained in the java.lang package, and subclasses of Throwable are contained in various packages.
• Errors related to GUI components are included in the java.awt package; numeric exceptions are included in the java.lang package because they are related to the java.lang.Number class.
• You can create your own exception classes by extending Throwable or a subclass of Throwable.

Object Oriented Programming module | PDF page 28 | lines 8-22

```
try {
int num=
reader.nextInt();
System.out.println("The number
is :" + num);
}
catch
(InputMismatchException e){
System.out.println("There is
type mis match :");
}
finally{
System.out.println("This part is
always done");
}} }
```

Object Oriented Programming module | PDF page 29 | lines 11-22

```
Using try and catch...
General form of try/catch:
Try{
//block of code to monitor for
errors
}
Catch(ExceptionType1 exOb ) {
//handler for ExceptionType1
}
Catch(ExceptionType2 exOb ) {
//handler for ExceptionType2
}
```

Question sources

Computer Science MCQ practice collection · Question 316

Question 204 | AM0317

Which independent Java declarations correctly create an int array? Select all that apply.

A. int[] numbers = {1, 2, 3};

B. int numbers[] = new int[3];

C. int numbers = new int[]{1, 2, 3};

D. int numbers[] = {1, 2, 3};

Hint

Check declared dimensions, initializer shape, fixed length and zero-based index bounds.

Answer A, B, D: int[] numbers = {1, 2, 3};; int numbers[] = new int[3];; int numbers[] = {1, 2, 3};

A and D use legal bracket placements with a brace initializer to create arrays containing 1, 2 and 3. B also declares an int array and allocates three elements, initially zero. C instead declares numbers as a scalar int but supplies an int-array value, producing a type mismatch. Both bracket placement styles and both allocation forms are allowed, so A, B and D are selected.

Why each choice is right or wrong

A - Correct: The int[] type and brace initializer create a three-element int array.

B - Correct: Brackets after the variable name and new int[3] are valid.

C - Incorrect: The scalar int variable cannot hold the int[] initializer value.

D - Correct: The name[] form with a brace initializer is valid.

Primary note proof

Study-note section OOP26

> int[] a=new int[5]; and int a[]={1,2,3}; are valid. int[] a={}; creates a zero-length array; explicit new is unnecessary in a declaration initializer. int a[],b; makes only a an array; int[] c,d; makes both arrays. Array objects have fixed length after creation, although the reference may be reassigned.

Study-note section OOP05

> Java fields and array elements receive defaults: numeric zero, char '\u0000', boolean false and references null.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 6 | lines 4-17

> Java has two basic data types
> 1. Primitive Data Types: are 8 in java
> (byte, short, int, long, float, double,
> char & Boolean)

2. Reference types: Arrays, classes, and interfaces, String)
• Note: Boolean type has a value true or false (there is no conversion b/n Boolean and other types).
• All non-primitive types are reference types, so all class types are reference types.
• All integers in Java are signed, Java doesn‟t support unsigned integers.

Question sources

Computer Science MCQ practice collection · Question 317

Question 205 | AM0318

Which method is automatically called when an object is created in Java?

A. start()

B. run()

C. main()

D. constructor

Hint

Check the class name, return-type absence and the constructors actually declared.

Answer D: constructor

The constructor is automatically called on object creation before new completes.

Why each choice is right or wrong

A - Incorrect: Not the automatic creation routine defined here

B - Incorrect: Not the automatic creation routine defined here

C - Incorrect: main is the program entry method, not every object's initialization

D - Correct: Matches automatic object initialization

Primary note proof

Study-note section OOP09

A constructor has the class’s exact name and no return type, not even void. It initializes the new object and can be overloaded or private.

Study-note section OOP03

Student s=new Student(); allocates and initializes an instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 12 | lines 24-47

Constructors
• Performs automatic initialization of objects. .:.
• Initializes an object immediately upon creation.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
• The default constructor automatically initializes all instance variables to zero and all object references to NULL.
• The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
• Once you define your own constructor, the default constructor is no longer used.
• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Question sources

Computer Science MCQ practice collection · Question 318

Question 206 | AM0319

What is the role of the super keyword in Java?

A. To refer to the current object instance.

B. To invoke a method from the superclass.

C. To define a class as final.

D. To allocate memory for an object.

Hint

Distinguish the current instance, parent context, hidden fields and static dispatch.

Answer B: To invoke a method from the superclass.

super.member may name a method or instance variable of the immediate superclass. This directly covers invoking a superclass method.

Why each choice is right or wrong

A - Incorrect: Current-object role is this rather than super

B - Correct: Matches superclass method access

C - Incorrect: final is the separate class modifier

D - Incorrect: new is the allocation operation

Primary note proof

Study-note section OOP16

> Java this denotes the current object; this.name=name distinguishes a field from a parameter. super.name selects a hidden superclass field; super.sound() directly requests the parent implementation. It does not create a second parent object.

Study-note section OOP07

> final prevents reassignment of a variable, overriding of a method or extension of a class.

Study-note section OOP03

> Student s=new Student(); allocates and initializes an instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 17 | lines 51-74

> The super keyword
> • Since encapsulation is a primary attribute of OOP, Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.
> • Super has two general forms.
> 1. Used to access parent class instance members.
> 2. Used to access parent class constructor.
> 1. Used to access parent class instance members.
> • super acts somewhat like this, except that it always refers to the superclass of the subclass in which it is used.
> • Super has the form: super.member Here, member can be either a method or an

instance variable.
• Super is most applicable to resolve member hiding.
• super( ) must always be the first statement executed inside a subclass‟ constructor.

Question sources

Computer Science MCQ practice collection · Question 319

Question 207 | AM0320

What does the static keyword mean in Java?

A. It denotes that a variable or method belongs to the superclass.

B. It indicates that a variable or method can only be accessed within the same class.

C. It allows a variable or method to be accessed without creating an instance of the class.

D. It specifies that a variable or method cannot be modified.

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer C: It allows a variable or method to be accessed without creating an instance of the class.

Static members belong to the class, and the static method is accessible by class name without an object. Visibility and immutability are separate properties.

Why each choice is right or wrong

A - Incorrect: Class membership does not mean belonging to the superclass

B - Incorrect: Private is the visibility restriction

C - Correct: Matches class access without creating an instance

D - Incorrect: final, not static, prevents modification

Primary note proof

Study-note section OOP07

static gives one field per loaded class, shared by its instances. If count starts at zero and every constructor increments it, three creations leave count=3. Prefer ClassName.count and ClassName.utility(). A static method has no current instance, but can still use instance members through an explicit object reference.

Study-note section OOP07

final prevents reassignment of a variable, overriding of a method or extension of a class.

Study-note section OOP08

For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 14 | lines 2-19

Static methods and Final Variables
• A static method can access only static data.
• A static method can call only other static data (Static methods & variables), but can not call a non-static data from it.
• Static method is belongs to the class and not to the object.
• A static method can be accessed directly by the class name and doesn‟t need any object.
• i.e ClassName.methodName(args)
• A static method cannot refer to "this" or "super" keywords in anyway
• But can be overloaded since they are resolved using static binding by compiler at compile time.

Object Oriented Programming module | PDF page 5 | lines 39-61

C. Class/Static variables
• Are declared with the static keyword in a class, but outside a method, constructor or a block.
• There would only be one copy of each class variable per class, regardless of how many objects are created from it.
• Are rarely used other than being declared as constants. (public static final)
• Are stored in the static memory.
• Static variables are created when the program starts and destroyed when the program stops.
• Visibility is similar to instance

variables. However, most static variables are declared public
• Default values are same as instance variables.
• Static variables can be accessed by calling with the class name ClassName.VariableName.

Question sources

Computer Science MCQ practice collection · Question 320

Question 208 | AM0321

What is the output of the following Java code snippet?

```
class MyClass {
public static void main(String[] args) {
String text = "Hello";
text += " World";
System.out.println(text);
}
}
```

A. Hello

B. World

C. Hello World

D. Compile error

Hint

Separate identity, content equality, immutable values and references that may be reassigned.

Answer C: Hello World.

OOP22 directly explains that text += " World" assigns a reference to a new concatenated result rather than mutating the original String. The variable therefore refers to "Hello World" before printing. String immutability does not prevent rebinding a non-final variable, so the code is valid and prints the combined text with its space.

Why each choice is right or wrong

A - Incorrect: The variable is rebound to the concatenated result, so it no longer prints only its initial text.

B - Incorrect: Concatenation retains Hello and appends the space and World; it does not replace the entire content with World.

C - Correct: The new concatenated String contains Hello, a space and World, and text refers to that result.

D - Incorrect: String immutability permits this reference reassignment; the code does not attempt to mutate the original String.

Primary note proof

Study-note section OOP22

> Java String objects are immutable. text += " World" assigns a reference to a new concatenated result and can print Hello World without mutating the original String object. final class String prevents subclassing; final alone does not make arbitrary objects immutable.

Study-note section OOP24

> Once a String operand is encountered, + concatenates in left-to-right groups: 1+2+"x" gives "3x", while "x"+1+2 gives "x12".

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 9 | lines 14-16

> 5. String Class:- Immutable
> Class created to manipulate
> strings(Sequence of chars).

Object Oriented Programming module | PDF page 10 | lines 15-26

> • Every Dog object will contain its
> own copies of the instance variables
> breed, age, and color.
> • To access these variables, you will
> use the dot (.) operator.
> • The dot operator links the name of
> the object with the name of an
> instance variable.
> • For example, to assign the age
> variable of myDog the value 10, you
> would use the following statement:
> myDog.age = 10; //This is called

Object Oriented Programming module | PDF page 4 | lines 21-34

> ✓ Operators: Are symbols that
> take one or more arguments
> (operands) and operates on
> them to produce a result. An
> operator performs a function
> on one (Unary), two (Binary),
> or three operands (Ternary).
> Java has one ternary operator,
> ?:, which is a short-hand if-
> else statement. E.g. c=x?w:s;

Operators can be Arithmetic,
Logical, assignment,
Relational, conditional,
Increment,

Question sources

Computer Science MCQ practice collection · Question 321

Question 209 | AM0322

Which access modifier allows a class member to be accessible only within the same package?

A. public

B. private

C. protected

D. Default (no modifier specified)

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer D: Default (no modifier specified)

No modifier/default is explicitly package-private. Other modifiers are narrower or wider than the package-only requirement.

Why each choice is right or wrong

A - Incorrect: World-visible, too broad

B - Incorrect: Class-only, too narrow

C - Incorrect: Also includes out-of-package subclasses

D - Correct: Exactly the package-private scope

Primary note proof

Study-note section OOP08

For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Study-note section OOP08

Across packages, a subclass accessing a protected instance member must do so through its own inherited context or an appropriate subclass-typed receiver, not any arbitrary superclass object. protected therefore does not mean “subclasses only”: unrelated classes in the same package also have access.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

Access Specifiers
Access level modifiers determine whether other classes can use a particular field or invoke a particular method.
Their are four types access levels:
1. No Modifier(Default): Visible to the package(source file), and it is called package-private
2. Private: Visible to the class only.
3. Public: Visible to the world.
4. Protected: Visible to the package and all subclasses.

Question sources

Computer Science MCQ practice collection · Question 322

Question 210 | AM0323

What is the output of the following Java code snippet?

```
class MyClass {
private int x = 5;

public int getX() {
return x;
}

public static void main(String[] args) {
MyClass obj = new MyClass();
obj.x = 10;
System.out.println(obj.getX());
}
}
```

A. 5

B. 10

C. Compile error

D. Runtime exception

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer B: 10

The write to obj.x is inside MyClass itself, even though main is static; private is class-based, not object-based. obj.x=10 updates the instance, and getX returns that field.

Why each choice is right or wrong

A - Incorrect: Old initializer is replaced by the assignment

B - Correct: Getter returns the updated field value 10

C - Incorrect: Same-class code may access private fields through obj

D - Incorrect: No exceptional operation is indicated by these class-local accesses

Primary note proof

Study-note section OOP08

> A class's own main method can access private fields of an instance of that class.

Study-note section OOP07

> A static method has no current instance, but can still use instance members through an explicit object reference.

Study-note section OOP11

> An accessor/getter returns information; a mutator/setter changes state.

Study-note section OOP-S03

> A later assignment overwrites an earlier value.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

> Access Specifiers
> Access level modifiers determine whether
> other classes can use a particular field or
> invoke a particular method.
> Their are four types access levels:
> 1. No Modifier(Default): Visible to
> the package(source file), and it is
> called package-private
> 2. Private: Visible to the class only.
> 3. Public: Visible to the world.
> 4. Protected: Visible to the package
> and all subclasses.

Object Oriented Programming module | PDF page 10 | lines 15-26

> • Every Dog object will contain its
> own copies of the instance variables

breed, age, and color.
• To access these variables, you will use the dot (.) operator.
• The dot operator links the name of the object with the name of an instance variable.
• For example, to assign the age variable of myDog the value 10, you would use the following statement: myDog.age = 10; //This is called

Question sources

Computer Science MCQ practice collection · Question 323

Question 211 | AM0324

What is the purpose of the toString() method in Java?

A. To convert an object to a string representation.

B. To compare two objects for equality.

C. To perform arithmetic calculations.

D. To access the superclass methods.

Hint

Separate identity, content equality, immutable values and references that may be reassigned.

Answer A: Return an object's textual representation.

OOP22 defines toString as giving a textual representation and distinguishes it from an equality test. A states that purpose. Comparing objects is the role of equals under the class's policy, while arithmetic and superclass method access are separate operations. The representation need not create a new copy of the original object.

Why each choice is right or wrong

A - Correct: A textual String representation of an object is the purpose explicitly assigned to toString in OOP22.

B - Incorrect: Object equality is handled by equals under the class's equality policy, not by toString.

C - Incorrect: Arithmetic calculations are not the defining contract of an object's textual representation method.

D - Incorrect: Superclass access uses super; it is not the purpose of toString.

Primary note proof

Study-note section OOP22

For object references, == tests identity; equals invokes the class's equality policy. Object.equals uses identity unless overridden; String.equals compares character content. Equal objects must have equal hashCode values when that contract is implemented. toString gives a textual representation, not an equality test.

Study-note section OOP16

super.name selects a hidden superclass field; super.sound() directly requests the parent implementation.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 10 | lines 62-72

Methods
• Are members of classes and do operations.
• Allows different classes to communicate each other.
• E.g. type name(parameter-list)
{ // body of method
//....
return value; // if type is not void

Object Oriented Programming module | PDF page 9 | lines 14-16

5. String Class:- Immutable Class created to manipulate strings(Sequence of chars).

Question sources

Computer Science MCQ practice collection · Question 324

Question 212 | AM0325

What is the purpose of the equals() method in Java?

A. To convert an object to a string representation.

B. To compare two objects for equality.

C. To perform arithmetic calculations.

D. To access the superclass methods.

Hint

Separate identity, content equality, immutable values and references that may be reassigned.

Answer B: Compare objects according to the class's equality policy.

OOP22 says equals invokes a class's equality policy; Object.equals uses identity unless overridden, while String.equals compares content. B is therefore correct without assuming every class implements value equality. The other choices describe textual representation, arithmetic or superclass access, none of which is equals' comparison purpose.

Why each choice is right or wrong

A - Incorrect: Textual representation is the purpose of toString, rather than equals.

B - Correct: equals compares objects using the class's policy, which may be identity or an overridden value/content policy.

C - Incorrect: equals supplies an equality result rather than being an arithmetic calculation mechanism.

D - Incorrect: Superclass access uses super and is distinct from the equality comparison performed by equals.

Primary note proof

Study-note section OOP22

> For object references, == tests identity; equals invokes the class's equality policy. Object.equals uses identity unless overridden; String.equals compares character content. Equal objects must have equal hashCode values when that contract is implemented. toString gives a textual representation, not an equality test.

Study-note section OOP16

> super.name selects a hidden superclass field; super.sound() directly requests the parent implementation.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 10 | lines 62-72

> Methods
> • Are members of classes and do
> operations.
> • Allows different classes to
> communicate each other.
> • E.g. type name(parameter-
> list)
> { // body of method
> //....
> return value; // if
> type is not void

Question sources

Computer Science MCQ practice collection · Question 325

Question 213 | AM0326

What is the correct way to create an instance of a class in Java?

A. MyClass obj = new MyClass();

B. MyClass obj = MyClass();

C. MyClass obj = create MyClass();

D. MyClass obj = instance MyClass();

Hint

Separate a reference declaration, object allocation, reference copying and changes to the shared object.

Answer A: MyClass obj = new MyClass();

Dog myDog = new Dog(); includes a declaration, new allocation, constructor parentheses and assignment. A follows precisely that form.

Why each choice is right or wrong

A - Correct: Matches the demonstrated object construction

B - Incorrect: Omission of new does not match construction syntax

C - Incorrect: create is not Java's allocation keyword

D - Incorrect: instance is not Java's allocation keyword

Primary note proof

Study-note section OOP03

> Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance.

Study-note section OOP09

> A constructor has the class's exact name and no return type, not even void. It initializes the new object and can be overloaded or private.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 8 | lines 43-55

> • a class declaration only creates a template; it does not create an actual object.
> • To actually create a Dog object, use the second statement.
> • Dog myDog; // Creates only the template

• myDog = new Dog(); //
create a Dog object called
myDog
• After this statement executes,
myDog will be an instance of
Dog.

Question sources

Computer Science MCQ practice collection · Question 326

Question 214 | AM0327

What is the purpose of the static keyword for a method in Java?

A. It indicates that the method can be overridden.

B. It allows the method to be accessed without creating an instance of the class.

C. It denotes that the method can only be called by the class itself.

D. It specifies that the method cannot be modified.

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer B: It allows the method to be accessed without creating an instance of the class.

Static methods can be called directly by class name without an object and cannot be overridden; a subclass declaration can hide them.

Why each choice is right or wrong

A - Incorrect: Contradicts the static non-overriding rule

B - Correct: Matches object-free class access

C - Incorrect: Visibility depends on access modifiers, not static alone

D - Incorrect: Immutability/non-overriding is not the same as prohibition on modifying the method

Primary note proof

Study-note section OOP07

static gives one field per loaded class, shared by its instances. If count starts at zero and every constructor increments it, three creations leave count=3. Prefer ClassName.count and ClassName.utility(). A static method has no current instance, but can still use instance members through an explicit object reference.

Study-note section OOP14

Constructors, final methods, static methods and private parent methods are not overridden.

Study-note section OOP08

For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 14 | lines 2-19

Static methods and Final Variables
• A static method can access only static data.
• A static method can call only other static data (Static methods & variables), but can not call a non-static data from it.
• Static method is belongs to the class and not to the object.
• A static method can be accessed directly by the class name and doesn‟t need any object.
• i.e ClassName.methodName(args)
• A static method cannot refer to "this" or "super" keywords in anyway
• But can be overloaded since they are resolved using static binding by compiler at compile time.

Object Oriented Programming module | PDF page 21 | lines 26-39

Rules of method overriding in Java..
• Argument list: The argument list of overriding (method of subclass) and Overridden (method of super class) method must be similar.
• The data types of the arguments and their sequence should exactly match.
• Method names: must be similar.
• private, static and final methods cannot be overridden as they are local to the class.
• Access Modifier of the overriding method cannot be more restrictive than the overridden method of parent class.

Question sources

Computer Science MCQ practice collection · Question 327

Question 215 | AM0328

What is the role of the this keyword in Java?

A. It refers to the current object instance.

B. It refers to the superclass of the current class.

C. It refers to the class that contains the current method.

D. It refers to the class from which the current class is derived.

Hint

Distinguish the current instance, parent context, hidden fields and static dispatch.

Answer A: this refers to the current object instance.

OOP16 defines this as the current object. In the main options that statement is A; it denotes an instance rather than its class, subclass or superclass. The source variant moves the same correct statement to C, so its labels must be reviewed separately. super is the distinct parent-context mechanism.

Why each choice is right or wrong

A - Correct: The current object instance is exactly what Java this denotes in ordinary instance context.

B - Incorrect: Superclass context is selected by super, not by the current-object reference this.

C - Incorrect: this refers to an instance, not to the class declaration containing a method.

D - Incorrect: The class from which the current class derives is its superclass, not the meaning of this.

Primary note proof

Study-note section OOP16

> Java this denotes the current object; this.name=name distinguishes a field from a parameter. super.name selects a hidden superclass field; super.sound() directly requests the parent implementation. It does not create a second parent object.

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 17 | lines 51-74

The super keyword
• Since encapsulation is a primary attribute of OOP, Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.
• Super has two general forms.
1. Used to access parent class instance members.
2. Used to access parent class constructor.
1. Used to access parent class instance members.
• super acts somewhat like this, except that it always refers to the superclass of the subclass in which it is used.
• Super has the form: super.member Here, member can be either a method or an instance variable.
• Super is most applicable to resolve member hiding.
• super( ) must always be the first statement executed inside a subclass‟ constructor.

Variant 1 | Alternative version 1

What is the role of the this keyword in Java?

A. It refers to the superclass of the current class.

B. It refers to the subclass of the current class.

C. It refers to the current object instance.

D. It refers to the class that contains the current method.

Answer C: this refers to the current object instance.

OOP16 defines this as the current object. This source variant places that definition at C, rather than main-option A. The superclass and subclass alternatives name type relationships, and the containing-class alternative denotes a type rather than the current instance.

Why each choice is right or wrong

A - Incorrect: Superclass context is associated with super, rather than the current-instance reference this.

B - Incorrect: A subclass is a type relationship; this denotes the current object, not a subclass.

C - Correct: This variant places the exact current-object-instance definition from OOP16 at label C.

D - Incorrect: The containing class is a type; this refers to the current object instance of a class.

Primary note proof

Study-note section OOP16

> Java this denotes the current object; this.name=name distinguishes a field from a parameter. super.name selects a hidden superclass field; super.sound() directly requests the parent implementation. It does not create a second parent object.

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent.

Question sources

Computer Science MCQ practice collection · Question 328

Computer Science MCQ practice collection · Question 342

Question 216 | AM0329

What is a constructor in Java?

A. A method that returns a value.

B. A method used for exception handling.

C. A special method used to initialize objects.

D. A method used to override a superclass method.

Hint

Check the class name, return-type absence and the constructors actually declared.

Answer C: A special method used to initialize objects.

Constructors automatically initialize objects, have the same name as their class and have no return type.

Why each choice is right or wrong

A - Incorrect: Contradicts the no-return-type rule

B - Incorrect: Exception handling uses try/catch

C - Correct: Matches automatic initialization of new objects

D - Incorrect: Constructors are not inherited overriding methods

Primary note proof

Study-note section OOP09

> A constructor has the class's exact name and no return type, not even void. It initializes the new object and can be overloaded or private.

Study-note section OOP-S05

> In Java's class-declaration terminology, constructors are not members and are not inherited.

Study-note section OOP30

try encloses guarded operations; a matching catch handles the thrown type or a subtype. Execution skips the remainder of the try when a throw occurs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 12 | lines 24-47

Constructors
• Performs automatic initialization of objects. .:.
• Initializes an object immediately upon creation.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
• The default constructor automatically initializes all instance variables to zero and all object references to NULL.
• The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
• Once you define your own constructor, the default constructor is no longer used.
• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Question sources

Computer Science MCQ practice collection · Question 329

Question 217 | AM0331

Assume an accessible inherited instance method is non-final, the subclass supplies the same name and parameter list with a compatible return type, and visibility is not reduced. Which descriptions fit the resulting method overriding? Select all that apply.

A. Creating multiple methods with the same name but different return types.

B. Creating multiple methods with the same name but different parameter lists.

C. Creating a new method with the same name and parameter list in the subclass.

D. Redefining a method in the subclass that already exists in the superclass.

Hint

Check inheritance, signature compatibility, visibility and eligibility for an override.

Answer C, D: Creating a new method with the same name and parameter list in the subclass.; Redefining a method in the subclass that already exists in the superclass.

Under the stated eligibility, signature, return and visibility conditions, the subclass method overrides the inherited method. C describes the same name and parameters, and D describes redefining the inherited superclass operation, so both describe this case. Different parameter lists denote overloading instead. Merely changing return types does not define overriding or a valid overload. The assumptions make the two overlapping valid descriptions explicit.

Why each choice is right or wrong

A - Incorrect: Different return types alone do not define overriding or distinguish valid overloads.

B - Incorrect: Different parameter lists describe overloading rather than this same-signature override.

C - Correct: The same-name, same-parameter subclass declaration is an override under the supplied conditions.

D - Correct: It redefines the eligible inherited superclass operation under the supplied conditions.

Primary note proof

Study-note section OOP14

> A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype. Visibility cannot become more restrictive, and checked exceptions cannot be broadened incompatibly.

Study-note section OOP12

> Overloaded methods share a name but have different parameter types, counts or type order. Return type alone is insufficient in Java and C++.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 11 | lines 55-67

> Method Overriding
> In a class hierarchy, when a method in a
> subclass has the same name and type
> signature as a method in its superclass, this
> is called method Overriding.
> • the method in the subclass is said to
> override the method in the
> superclass.
> • The Method in the super class is

known as an overridden method
which is called within a subclass.
• The version of the method defined
by the superclass will be hidden.

Object Oriented Programming module | PDF page 21 | lines 26-39

Rules of method overriding in Java..
• Argument list: The argument list of
overriding (method of subclass) and
Overridden (method of super class)
method must be similar.
• The data types of the arguments and
their sequence should exactly match.
• Method names: must be similar.
• private, static and final methods cannot
be overridden as they are local to the
class.
• Access Modifier of the overriding
method cannot be more restrictive than
the overridden method of parent class.

Question sources

Computer Science MCQ practice collection · Question 331

Question 218 | AM0333

What is polymorphism in Java?

A. The ability to create multiple objects of the same class.

B. The process of converting an object to a string representation.

C. The ability of an object to take on many forms through inheritance.

D. The process of comparing two objects for equality.

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer C: The ability of an object to take on many forms through inheritance.

Polymorphism means many forms, typically illustrated by a superclass reference pointing at child objects; inheritance supplies those related forms.

Why each choice is right or wrong

A - Incorrect: Several objects of the same class is multiplicity of instances

B - Incorrect: Object-to-string conversion is unrelated

C - Correct: Matches many forms through a hierarchy

D - Incorrect: Equality comparison is unrelated

Primary note proof

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Study-note section OOP15

> For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile.

Study-note section OOP22

> For object references, == tests identity; equals invokes the class's equality policy. Object.equals uses identity unless overridden; String.equals compares character content. Equal objects must have equal hashCode values when that contract is implemented. toString gives a textual representation, not an equality test.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 25 | lines 17-37

> • Polymorphism is the ability of an object to take on many forms.
> • The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
> • Any Java object that can pass more than one IS-A test is considered to be polymorphic.
> • In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.
> • Polymorphism is derived from 2 Greek words: poly and morphs.
> • The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
> • polymorphism is the capability of an action or method to do different things based on the object that it is acting upon.

Question sources

Computer Science MCQ practice collection · Question 333

Question 219 | AM0334

Which object-oriented principle promotes the concept of reusing code and creating relationships between classes?

A. Encapsulation

B. Inheritance

C. Polymorphism

D. Abstraction

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer B: Inheritance

The inheritance section explicitly links reuse to creating a new class that inherits Person without changing Person, forming the parent/child relationship.

Why each choice is right or wrong

A - Incorrect: Wraps and hides data rather than deriving related classes

B - Correct: Directly supplies reuse through inherited relationships

C - Incorrect: Uses varying behavior through types rather than creating the reuse relationship

D - Incorrect: Suppresses details rather than acquiring reusable members

Primary note proof

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects. Prefer a precise mechanism over a vague slogan: wrapping fields and methods points to encapsulation; showing what an operation does while hiding how points to abstraction.

Study-note section OOP02

> Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Study-note section OOP13

> Use inheritance when a subtype can satisfy the parent's contract: Dog is an Animal. For shared Pet.name and speak(), put common state/contract in Pet, specialized fetch() in Dog, and appropriate speak() implementations in Dog and Cat. Composition represents has-a, such as Car containing Engine.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 46-56

5. Inheritance:
• It is the way in which objects of one class get the properties of objects of another class(data &methods).
• For example, Person, employee and retiree
• Inheritance provides the idea of reusability - the Person class can be reused without making changes to it, because another class can be made to inherit from it.

Object Oriented Programming module | PDF page 15 | lines 23-53

What is Inheritance?
• Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{

```
.....}
class Sub extends Super{ }
```

Question sources

Computer Science MCQ practice collection · Question 334

Question 220 | AM0335

What is the purpose of encapsulation in object-oriented programming?

A. To create relationships between classes

B. To allow classes to inherit properties and behaviors

C. To hide internal implementation details and expose a public interface

D. To provide different implementations of a method based on the object's type

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer C: To hide internal implementation details and expose a public interface

The hidden-data part of encapsulation exposes access through methods, which is the public interaction boundary described by C. A/B describe inheritance and D describes polymorphic overriding.

Why each choice is right or wrong

A - Incorrect: Describes class relationships generally

B - Incorrect: Describes inheritance specifically

C - Correct: Matches hidden data accessed through methods

D - Incorrect: Describes different behavior based on receiver type

Primary note proof

Study-note section OOP02

> Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller. These ideas overlap: a private balance plus deposit(amount) can demonstrate all three, depending on what the question emphasizes.

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 35-45

𝟒. Encapsulation:
• means the wrapping up of data and methods into a single unit (a class).
• It also means that the data inside a class is hidden from everything outside the class.
• The data can only be accessed by invoking the methods of the class.
• It is placing the data and the functions that work on that data in the same place.

Object Oriented Programming module | PDF page 3 | lines 27-33

𝟐. Abstraction:
• provides only essential information to the outside world.
• Or hiding their background details, i.e., to represent the needed information in program without presenting the details.

Question sources

Computer Science MCQ practice collection · Question 335

Question 221 | AM0336

Which object-oriented principle allows an object to exhibit different behaviors based on its data type?

A. Encapsulation

B. Inheritance

C. Polymorphism

D. Abstraction

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer C: Polymorphism

An operation can behave differently for different subclasses, with method behavior determined by the object acted upon.

Why each choice is right or wrong

A - Incorrect: Groups and hides data

B - Incorrect: Creates the subtype relation but does not name varying operation behavior

C - Correct: Matches behavior varying with object type

D - Incorrect: Shows essentials while suppressing details

Primary note proof

Study-note section OOP02

> Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Study-note section OOP15

> For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 58-64

> 2. Polymorphism:
> • The word polymorphism means the ability to take more than one form.
> • In terms of the OOP, this means that a particular operation may behave differently for different sub-classes of the same class.

Object Oriented Programming module | PDF page 25 | lines 17-37

> • Polymorphism is the ability of an object to take on many forms.
> • The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
> • Any Java object that can pass more than one IS-A test is considered to be polymorphic.
> • In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class

Object.
• Polymorphism is derived from 2 Greek words: poly and morphs.
• The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
• polymorphism is the capability of an action or method to do different things based on the object that it is acting upon.

Question sources

Computer Science MCQ practice collection · Question 336

Question 222 | AM0337

What is the purpose of abstraction in object-oriented programming?

A. To create relationships between classes

B. To allow classes to inherit properties and behaviors

C. To hide internal implementation details and expose a public interface

D. To provide different implementations of a method based on the object's type

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer C: To hide internal implementation details and expose a public interface

Abstraction represents needed/essential information while hiding background details, matching a public-facing interface to hidden implementation.

Why each choice is right or wrong

A - Incorrect: Describes relationships rather than the essential view

B - Incorrect: Describes inheritance

C - Correct: Matches essential information with background details hidden

D - Incorrect: Describes polymorphic differing implementations

Primary note proof

Study-note section OOP02

Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller. These ideas overlap: a private balance plus deposit(amount) can demonstrate all three, depending on what the question emphasizes.

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 27-33

> 0. Abstraction:
> • provides only essential information
> to the outside world.
> • Or hiding their background details,
> i.e., to represent the needed
> information in program without
> presenting the details.

Question sources

Computer Science MCQ practice collection · Question 337

Question 223 | AM0338

Which Java keyword declares that one class inherits from another class?

A. super

B. extends

C. implements

D. this

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer B: extends

A class declaration uses extends to name its direct superclass, as in class Dog extends Animal. A class implements interface contracts using implements, which is a different declaration relationship. super selects parent context inside a class, while this denotes the current instance. Restricting the question to class-to-class inheritance makes extends the unique keyword among these choices.

Why each choice is right or wrong

A - Incorrect: Refers to superclass members/constructor

B - Correct: Matches class Sub extends Super

C - Incorrect: Used for interface implementation instead

D - Incorrect: Refers to current-instance context, not an inheritance declaration

Primary note proof

Study-note section OOP13

class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent. A class extends one direct class and can implement multiple interfaces.

Study-note section OOP16

Java this denotes the current object; this.name=name distinguishes a field from a parameter. super.name selects a hidden superclass field; super.sound() directly requests the parent implementation.

Study-note section OOP19

A class can extend a superclass and implement zero or more interfaces. An interface can extend several interfaces.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 15 | lines 23-53

What is Inheritance?
• Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends

keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Object Oriented Programming module | PDF page 31 | lines 9-30

• It provides total abstraction;
• Means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default.
• A class that implement interface must implement all the methods declared in the interface.
• If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
Rules of Interface
• Variables declared in an interface are public, static & final by default.
• Java does not allow you to extend more than one class, However you can implement more than one interfaces in your class.
• class implements interface but an interface extends another interfaces.

Question sources

Computer Science MCQ practice collection · Question 338

Question 224 | AM0339

Which Java annotation requests compiler checking that a method overrides or implements an inherited method contract?

A. super

B. extends

C. implements

D. @Override

Hint

Check inheritance, signature compatibility, visibility and eligibility for an override.

Answer D: @Override

@Override asks the compiler to verify that a declaration satisfies overriding rules. Java does not have an override keyword, and the annotation does not create an override by itself; the declaration must still meet the inherited-contract conditions. super selects parent context, extends declares class inheritance, and implements declares interface relationships. Correcting the requested construct to an annotation preserves the intended override-checking concept.

Why each choice is right or wrong

A - Incorrect: super selects parent members or construction rather than checking a declaration's override status.

B - Incorrect: extends establishes class inheritance rather than annotating a method.

C - Incorrect: implements establishes an interface relationship rather than checking one method declaration.

D - Correct: @Override requests compiler verification of the overriding/implementation relationship.

Primary note proof

Study-note section OOP14

> A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype. Visibility cannot become more restrictive, and checked exceptions cannot be broadened incompatibly. @Override requests compiler checking; Java has no override keyword. Constructors, final methods, static methods and private parent methods are not overridden.

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent. A class extends one direct class and can implement multiple interfaces.

Study-note section OOP16

> super.name selects a hidden superclass field; super.sound() directly requests the parent implementation.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 11 | lines 55-67

> Method Overriding
> In a class hierarchy, when a method in a
> subclass has the same name and type
> signature as a method in its superclass, this
> is called method Overriding.
> • the method in the subclass is said to
> override the method in the
> superclass.

• The Method in the super class is known as an overridden method which is called within a subclass.
• The version of the method defined by the superclass will be hidden.

Object Oriented Programming module | PDF page 15 | lines 23-53

What is Inheritance?
• Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications.
• Inheritance is the process where one class acquires the properties (methods and fields) of another.
• A class that is inherited is called a superclass.
• The class that does the inheriting is called a subclass.
• OR: The class which inherits the properties of other is known as subclass (derived class, child class) and
• The class whose properties are inherited is known as superclass (base class, parent class).
• Therefore, a subclass is a specialized version of a superclass.
• It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.
• The extends keyword: is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.
• Sytax:
class Super{
.....}
class Sub extends Super{ }

Object Oriented Programming module | PDF page 21 | lines 26-39

Rules of method overriding in Java..
• Argument list: The argument list of overriding (method of subclass) and Overridden (method of super class) method must be similar.
• The data types of the arguments and

their sequence should exactly match.
• Method names: must be similar.
• private, static and final methods cannot be overridden as they are local to the class.
• Access Modifier of the overriding method cannot be more restrictive than the overridden method of parent class.

Question sources

Computer Science MCQ practice collection · Question 339

Question 225 | AM0341

Which object-oriented principle promotes the concept of grouping related properties and behaviors into a single unit?

A. Encapsulation

B. Inheritance

C. Polymorphism

D. Abstraction

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer A: Encapsulation

The stem repeats the definition of grouping data and methods into a single class unit.

Why each choice is right or wrong

A - Correct: Direct wrapping-up/grouping principle

B - Incorrect: Acquires another class's members

C - Incorrect: Many behavior forms

D - Incorrect: Essential-information view rather than grouping

Primary note proof

Study-note section OOP02

Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller. These ideas overlap: a private balance plus deposit(amount) can demonstrate all three, depending on what the question emphasizes.

Study-note section OOP02

Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 3 | lines 35-45

꣆. Encapsulation:
• means the wrapping up of data and methods into a single unit (a class).
• It also means that the data inside a class is hidden from everything outside the class.
• The data can only be accessed by invoking the methods of the class.
• It is placing the data and the functions that work on that data in the same place.

Question sources

Computer Science MCQ practice collection · Question 341

Question 226 | AM0343

Which object-oriented concept is used to create objects based on a predefined blueprint or class?

A. Polymorphism

B. Inheritance

C. Encapsulation

D. Instantiation

Hint

Separate a reference declaration, object allocation, reference copying and changes to the shared object.

Answer D: Instantiation

A class is the template and an object its instance; new creates that actual object. Instantiation names creating the instance from the template.

Why each choice is right or wrong

A - Incorrect: Many forms rather than creation from a blueprint

B - Incorrect: Deriving classes rather than objects

C - Incorrect: Grouping data/methods rather than object creation

D - Correct: Matches creating the actual instance from a class template

Primary note proof

Study-note section OOP03

> Student s; declares a reference variable; it creates no Student object. Student s=new Student(); allocates and initializes an instance.

Study-note section OOP02

> Encapsulation groups related state and operations and controls access.

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 8 | lines 43-55

> • a class declaration only creates a template; it does not create an actual object.
> • To actually create a Dog object, use the second statement.
> • Dog myDog; // Creates only the template
> • myDog = new Dog(); // create a Dog object called myDog
> • After this statement executes, myDog will be an instance of Dog.

Object Oriented Programming module | PDF page 8 | lines 4-20

> 2.1. Fundamentals of Classes
> • A class is a blue print from which individual objects are created.
> • Class is a template for an object, and an object is an instance of a class
> • Class is the logical construct upon which the entire Java language is built because it defines the shape and nature of an object.
> • As such, the class forms the basis for object-oriented programming in Java.
> • Classes usually consist of two things:

instance variables and methods
• Any concept you wish to implement
in a Java program must be
encapsulated within a class.

Question sources

Computer Science MCQ practice collection · Question 343

Question 227 | CS0100

Which one of the following is true regarding constructors in Java?

A. Constructors are invoked to create objects without using the new operator

B. Constructors can have a return type

C. A constructor can have a different name from its class name

D. Constructors play the role of initializing objects

Hint

Check the class name, return-type absence and the constructors actually declared.

Answer D: Constructors play the role of initializing objects

Constructors initialize objects automatically during new, share the class name and have no return type.

Why each choice is right or wrong

A - Incorrect: Contradicts the described new/constructor creation sequence for ordinary construction

B - Incorrect: Contradicts no return type, not even void

C - Incorrect: Contradicts same-class-name rule

D - Correct: Matches automatic initialization

Primary note proof

Study-note section OOP09

A constructor has the class's exact name and no return type, not even void. It initializes the new object and can be overloaded or private.

Study-note section OOP03

Student s=new Student(); allocates and initializes an instance.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 12 | lines 24-47

Constructors
• Performs automatic initialization of objects. .:.
• Initializes an object immediately upon creation.
• Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
• The default constructor automatically initializes all instance variables to zero and all object references to NULL.
• The default constructor is often sufficient for simple classes, but it usually won‟t do for more sophisticated ones.
• Once you define your own constructor, the default constructor is no longer used.
• Syntax:
• Have the same name with their class name.
• Have not return type, not even void.

Question sources

Computer Science Exit Exam practice collection · Question 100

Question 228 | CS0101

If you want to make methods and data fields of classes accessible from any class in any package, which access modifier is appropriate?

A. protected

B. default

C. public

D. private

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer C: public

Public means visible to the world, including unrelated classes in other packages.

Why each choice is right or wrong

A - Incorrect: Only package and subclasses, not every class

B - Incorrect: Only the package

C - Correct: World-visible, satisfying all classes/packages

D - Incorrect: Only the declaring class

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 13 | lines 62-73

> Access Specifiers
> Access level modifiers determine whether
> other classes can use a particular field or
> invoke a particular method.
> Their are four types access levels:
> 1. No Modifier(Default): Visible to
> the package(source file), and it is
> called package-private
> 2. Private: Visible to the class only.
> 3. Public: Visible to the world.
> 4. Protected: Visible to the package
> and all subclasses.

Question sources

Computer Science Exit Exam practice collection · Question 101

Question 229 | CS0102

Which one of the following is NOT correct about function overloading in C++?

A. Two or more overloaded functions may differ in their number of parameters

B. Two or more overloaded functions may differ in data types of their parameters

C. Two or more overloaded functions should have the same function name

D. Two or more overloaded functions may differ only in their return types

Hint

Compare names and parameter signatures before considering return types or access modifiers.

Answer D: Return type alone cannot distinguish C++ overloads.

OOP12 states that overloaded methods/functions share a name and differ in parameter types, counts or order. It expressly rejects return type alone as sufficient in Java and C++. D is therefore the incorrect overloading claim. A, B and C describe valid differences or the shared-name requirement rather than violating the overload rule.

Why each choice is right or wrong

A - Incorrect: Different parameter counts can distinguish overloads, so this is a correct overloading statement.

B - Incorrect: Different parameter data types can distinguish overloads, so this is a correct overloading statement.

C - Incorrect: Overloads share a function name, so this statement correctly describes their naming requirement.

D - Correct: This is the false claim: return type alone cannot distinguish C++ overloads with otherwise identical signatures.

Primary note proof

Study-note section OOP12

> Overloaded methods share a name but have different parameter types, counts or type order. Return type alone is insufficient in Java and C++.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Object Oriented Programming module | PDF page 11 | lines 7-18

> Method Overloading
> • Is a process of having the same name for different methods as long as their parameter list or data types in the parameter list is different?
> • The return type alone is insufficient to distinguish two versions of a method.
> • Parameters are key identifiers of OM.
> • Is one of the ways that Java supports polymorphism

Question sources

Computer Science Exit Exam practice collection · Question 102

Question 230 | NS0219

Which purposes apply to Java interfaces? “Blueprint” means an operation contract, “state and behavior” in B means complete per-instance state and implementation, and “multiple inheritance” means type inheritance. Select all that apply.

A. To create multiple constructors for a class

B. To define the state and behavior of an object

C. To define a blueprint of a class

D. To achieve multiple inheritance

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer C, D: To define a blueprint of a class; To achieve multiple inheritance

An interface gives implementing classes an operation contract, so the defined blueprint purpose is valid. A class can implement multiple interfaces, supporting multiple type inheritance. Interfaces have no constructors or per-instance fields, so they do not create a class’s constructors or define an object’s complete state and implementation. With those meanings explicit, C and D are valid purposes and the other two are excluded.

Why each choice is right or wrong

A - Incorrect: Interfaces have no constructors and do not create class constructors.

B - Incorrect: The complete per-instance state and implementation belong to classes, not interface capability contracts.

C - Correct: The blueprint is an operation contract for implementing classes.

D - Correct: Multiple implemented interfaces provide multiple type inheritance.

Primary note proof

Study-note section OOP19

> interface Payable{double getPaymentAmount();} declares an abstract public operation. class Invoice implements Payable must provide public double getPaymentAmount(), perhaps returning getQuantity()*getPricePerItem(). A class can extend a superclass and implement zero or more interfaces. An interface can extend several interfaces. No interface instance fields or constructors exist; fields are implicitly public static final.

Study-note section OOP20

> Use an abstract class for shared state, construction and common implementation; use an interface for a capability across unrelated classes and multiple type inheritance.

Question sources

Computer Science Exit Exam practice collection · Question 104

Question 231 | NS0292

Consider this C++17 class and statements inside a valid main function:
struct Meter {
int reading = 0;
void set(int reading) { this->reading = reading; }
static int twice(int x) { return 2 * x; }
};
Meter first, second;
first.set(4);
second.set(9);
int doubled = Meter::twice(3);
Which statement correctly describes the implicit object context during second.set(9) and the call to twice?

A. In set, this points to second; twice has no this pointer.

B. In set, this points to first; twice has no this pointer.

C. In set, this points to second; twice reuses second as its implicit object.

D. The hidden object argument is passed to the reading data member rather than to set.

Hint

Use the stated language rules and identify whether a member call has an implicit current object.

Answer A: In set, this points to second; twice has no this pointer.

Each non-static set call has an implicit object: first for first.set(4) and second for second.set(9). During the latter call, this therefore points to second, and this->reading selects that object's field. The static twice function has no current object or this pointer, regardless of which instance method ran most recently. Data members themselves do not receive hidden call arguments.

Why each choice is right or wrong

A - Correct: second is the receiver of second.set(9); the static twice call has no implicit object.

B - Incorrect: The earlier call on first does not make first the receiver of the later call on second.

C - Incorrect: twice is static and does not acquire a this pointer from the most recent instance call.

D - Incorrect: The implicit object belongs to the non-static function call; the reading data member does not receive a hidden argument.

Primary note proof

Study-note section OOP17

> In ordinary C++17 non-static member functions, this is a pointer to the object on which the function is called; this->field selects a member. It is often explained as an implicit object argument to a member function call. It is not an argument passed to member variables, and a static member function has no this pointer.

Question sources

Computer Science Exit Exam · 2018 · Question 9

Question 232 | NS0300

What will be the output of the following program segment?
public class Calculation {

public static void main(String[] args) {
int sum = 0;
for(int j = 1; j<=10; j++) {
sum = sum + j;
}
System.out.println(+sum);
}

A. 55

B. 44

C. 33

D. 66

Hint

Trace condition tests, control transfers and printing as separate steps.

Answer A: 55 for the shown program segment.

The shown program segment initializes sum to zero and adds j for all values 1 through 10 inclusive. Its numerical result is 55. The unary plus in println(+sum) preserves that value. This is a program segment; a complete compilation unit also requires the outer class's closing brace.

Why each choice is right or wrong

A - Correct: The ten inclusive additions produce 55; unary plus leaves that integer unchanged.

B - Incorrect: 44 is not the sum of the stated inclusive sequence 1 through 10.

C - Incorrect: 33 does not match all ten additions performed by the shown loop.

D - Incorrect: 66 would correspond to including an additional 11, which this condition excludes.

Primary note proof

Study-note section OOP25

> Count executions independently from printing.

Study-note section OOP25

> A loop summing j=1..10 yields 55.

Study-note section OOP-S03

Prefix ++ and -- update a variable before supplying its value; postfix forms supply the old value before the update. Multiplication, division and remainder bind before addition and subtraction; same-level binary arithmetic groups left to right. Integer division truncates toward zero. Unary plus preserves its numeric operand’s value.

Question sources

Computer Science Exit Exam · 2018 · Question 17

Question 233 | NS0318

Which OOP principle specifically defines the boundary that groups an object's state with its methods and controls clients' access to that state?

A. Abstraction

B. Inheritance

C. Polymorphism

D. Encapsulation

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer D: Encapsulation.

Encapsulation groups related state and operations and controls access to them. That grouping and access boundary is the mechanism requested. Abstraction emphasizes the essential public view while omitting irrelevant details; it can coexist with encapsulation but does not name this specific boundary. Inheritance forms subtype relationships, and polymorphism supplies appropriate behavior through a common type.

Why each choice is right or wrong

A - Incorrect: Abstraction concerns the essential view presented to callers, rather than the specific grouping/access boundary requested.

B - Incorrect: Inheritance forms a subtype relationship rather than a state-access boundary.

C - Incorrect: Polymorphism provides behavior through common types rather than defining this grouping and access control.

D - Correct: Encapsulation specifically groups state with operations and controls client access.

Primary note proof

Study-note section OOP02

> Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller. These ideas overlap: a private balance plus deposit(amount) can demonstrate all three, depending on what the question emphasizes.

Study-note section OOP02

> Inheritance forms a subtype relationship and may reuse implementation. Polymorphism lets code using a common type invoke behavior appropriate to different concrete objects.

Question sources

Computer Science Exit Exam · 2018 · Question 43

Question 234 | NS0335

Consider the following code snippet
class Animal { void sound() { System.out.println("Animal sound"); } }
class Dog extends Animal { void sound() { System.out.println("Bark!"); } }
Which OOP concept is illustrated when Dog provides its own version of sound?

A. Method overriding

B. Method overloading

C. Abstraction

D. Encapsulation

Hint

Check inheritance, signature compatibility, visibility and eligibility for an override.

Answer A: Method overriding.

Dog extends Animal and declares its own sound() with the same name, no parameters and the same void result. Neither method is static, private or final in the shown code. Dog therefore overrides the inherited instance method, providing subtype-specific behavior. No changed parameter list exists to make this an overload.

Why each choice is right or wrong

A - Correct: Dog supplies the same eligible inherited sound() signature with its own behavior, which is overriding.

B - Incorrect: Overloading requires a different parameter list; both sound methods have the same no-argument signature.

C - Incorrect: Abstraction is broader essential-interface design, not the specific same-signature subclass mechanism shown.

D - Incorrect: Encapsulation concerns grouped state/access control rather than this replacement of inherited method behavior.

Primary note proof

Study-note section OOP14

> A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype. Visibility cannot become more restrictive, and checked exceptions cannot be broadened incompatibly. @Override requests compiler checking; Java has no override keyword. Constructors, final methods, static methods and private parent methods are not overridden.

Study-note section OOP12

> Overloaded methods share a name but have different parameter types, counts or type order. Return type alone is insufficient in Java and C++.

Study-note section OOP02

Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller.

Question sources

Computer Science Exit Exam · 2018 · Question 65

Question 235 | NS0338

A member declared in package p must be usable by an unrelated helper in p and by a subclass in package q through its inherited context. Unrelated classes outside p must be excluded. Which access category fits these requirements?

A. Public

B. Default

C. Private

D. Protected

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer D: Protected, with qualified subclass access across packages.

Protected permits same-package access, including an unrelated helper, and qualified subclass access from another package. It does not grant access to unrelated clients in other packages. Public violates that exclusion, package/default access cannot reach the other-package subclass, and private cannot admit the unrelated same-package helper. For an instance member, the cross-package receiver restriction remains part of the protected rule.

Why each choice is right or wrong

A - Incorrect: Public would also admit the unrelated other-package clients that the stem excludes.

B - Incorrect: Default/package access would exclude the subclass in q.

C - Incorrect: Private would exclude the unrelated helper in p.

D - Correct: Protected provides both requested contexts while retaining the stated other-package exclusion.

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Study-note section OOP08

> Across packages, a subclass accessing a protected instance member must do so through its own inherited context or an appropriate subclass-typed receiver, not any arbitrary superclass object. protected therefore does not mean “subclasses only”: unrelated classes in the same package also have access.

Question sources

Computer Science Exit Exam · 2018 · Question 69

Question 236 | NS0345

Consider these Java 17 classes:
class Employee {
public final double calculateTax() { return 100.0; }
}
class Manager extends Employee {
// Insert exactly one candidate declaration here.
}
Which candidate fails compilation because it attempts to override the inherited final method? Evaluate each candidate independently.

A. public double calculateTax(double income) { return income * 0.1; }

B. public double calculateTax() { return 200.0; }

C. private double calculateTax(int category) { return category * 10.0; }

D. public static double calculateTax(String band) { return 0.0; }

Hint

Check inheritance, signature compatibility, visibility and eligibility for an override.

Answer B: public double calculateTax() { return 200.0; }

Employee's no-argument calculateTax method is final, so Manager cannot provide another instance method with that same signature. B attempts exactly that override and fails compilation. A and C have different parameter types and therefore declare overloads. D also has a different parameter list; its static declaration is a separate overload, not a replacement for the inherited no-argument instance method. The final restriction does not prohibit every reuse of the name.

Why each choice is right or wrong

A - Incorrect: The double parameter creates a different signature, so this overload does not override the final no-argument method.

B - Correct: This has the inherited no-argument signature and attempts to override the final method.

C - Incorrect: The int parameter creates an independent overload; its private access does not reduce the inherited method's visibility.

D - Incorrect: The String parameter creates a distinct static overload and leaves the inherited no-argument method unchanged.

Primary note proof

Study-note section OOP14

> Constructors, final methods, static methods and private parent methods are not overridden.

Study-note section OOP12

> Overloaded methods share a name but have different parameter types, counts or type order. Return type alone is insufficient in Java and C++.

Study-note section OOP07

> final prevents reassignment of a variable, overriding of a method or extension of a class.

Question sources

Computer Science Exit Exam · 2018 · Question 78

Question 237 | NS0632

A class has a constructor: public Circle(double r) { this.radius = r; }. What happens when you write: Circle c = new Circle(); ?

Hint

Check the class name, return-type absence and the constructors actually declared.

Answer: Compilation fails unless Circle also declares an accessible no-argument constructor.

Declaring Circle(double r) prevents Java from supplying the implicit default constructor. new Circle() requests a no-argument constructor, not the existing one-argument signature. OOP09 therefore predicts a compilation error unless another accessible no-argument constructor has been provided. Passing a double argument would instead target the shown constructor.

Primary note proof

Study-note section OOP09

> If a class declares no constructor, Java supplies a default constructor. If Circle(double r) is declared, new Circle() fails unless a no-argument constructor is also declared.

Question sources

Object-Oriented Programming review questions · Question 1

Question 238 | NS0633

Which of the following is TRUE about instance variables that are not explicitly initialized?

Hint

Locate the declaration scope and check default initialization versus definite assignment.

Answer: Uninitialized instance fields receive type defaults: numeric zero, false, null, or the zero char.

Instance variables are fields, so OOP05 gives them automatic defaults before explicit initialization. Numeric fields receive zero, boolean fields false, references null and char the zero code unit. This does not authorize reading an unassigned local variable: locals obey a separate definite-assignment requirement rather than receiving these field defaults.

Primary note proof

Study-note section OOP05

> Java fields and array elements receive defaults: numeric zero, char '\u0000', boolean false and references null. A local int t; followed by printing t is a compilation error.

Question sources

Object-Oriented Programming review questions · Question 2

Question 239 | NS0634

What is the output of: static int count = 0; in a class, after creating 3 objects each incrementing count++ in their constructor?

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer: The shared count becomes 3; printing it would output 3.

static makes count a single field shared by the instances, not three independent counters. Starting at zero, each of the three constructor increments adds one to that same field, leaving three. The declaration itself prints nothing; 3 is the stored value and the result if the question's output means subsequently printing count.

Primary note proof

Study-note section OOP07

> static gives one field per loaded class, shared by its instances. If count starts at zero and every constructor increments it, three creations leave count=3. Prefer ClassName.count and ClassName.utility().

Study-note section OOP-S02

> A print call emits its value without inserting a separator or ending the line.

Question sources

Object-Oriented Programming review questions · Question 3

Question 240 | NS0635

You declare: int number; inside a method and try to print it without assigning a value. What happens?

Hint

Locate the declaration scope and check default initialization versus definite assignment.

Answer: Compilation fails because the local variable has not been definitely assigned.

number is declared inside a method, making it a local rather than an instance or static field. OOP05 requires a local to be definitely assigned before use. Printing it is a read, so the compiler rejects that read. Java does not substitute field-style zero or permit an arbitrary garbage output for the local.

Primary note proof

Study-note section OOP05

> Locals occur in methods, constructors or blocks; they must be definitely assigned before use and cannot have public/private/protected modifiers.

Study-note section OOP05

> Java fields and array elements receive defaults: numeric zero, char '\u0000', boolean false and references null. A local int t; followed by printing t is a compilation error.

Question sources

Object-Oriented Programming review questions · Question 4

Question 241 | NS0636

In UML class diagrams, how is an instance shown differently from a class?

Hint

Read the notation and distinguish a class description from a particular instance.

Answer: An instance uses an underlined instanceName:ClassName label, often with slot values.

OOP04 gives an underlined label such as sara:Student for one particular object. A class box describes the common type with its name, attributes and operations; an instance notation can show actual slot values. Underlining static class features is a separate UML convention, so the full instanceName:ClassName pattern identifies the object here.

Primary note proof

Study-note section OOP04

> A UML class box normally contains name, attributes and operations. Visibility uses + public, - private and # protected. A field can be written name:String and an operation getName():String.

Study-note section OOP04

> An instance is conventionally labeled with an underlined instanceName:ClassName, such as sara:Student. Class notation describes a type; instance notation describes one particular object and may show its slot values. Underlining static class features has a different purpose; read the whole notation rather than assuming every underline means the same thing.

Question sources

Object-Oriented Programming review questions · Question 5

Question 242 | NS0637

Under Java 17, what is the key restriction on an explicit this(...) invocation inside a constructor?

Hint

Trace constructor delegation and initialization in the specified Java version.

Answer: It must be the first constructor statement and delegates to another constructor in the same class.

Under Java 17, an explicit this(...) invocation must be the first constructor statement and delegates to another constructor of the same class. The constructor cannot also explicitly invoke super(...); the delegation chain reaches superclass construction through another constructor. The version qualification matters because later Java versions permit a restricted prologue, so the Java 17 rule should not be presented as timeless.

Primary note proof

Study-note section OOP10

> Under Java 17 rules, an explicit this(args) or super(args) invocation must be the first constructor statement. this invokes another constructor in the same class; super invokes the immediate superclass constructor. A constructor cannot explicitly invoke both. If there is no explicit delegation, an implicit super() is inserted; a matching accessible no-argument parent constructor must exist.

Study-note section OOP10

> Java 25 permits restricted statements before explicit constructor invocation; the older first-statement rule must be version qualified.

Question sources

Object-Oriented Programming review questions · Question 6

Question 243 | NS0638

A class has: public static final int MAX = 100; Can a subclass change this value?

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer: No. final prevents assigning a new value to that MAX field.

MAX is one static field whose initialized value is 100. final prevents a subclass or other client from assigning a replacement value to that field. A subclass could declare a separate same-name hidden field, but that would not change the inherited constant. Public controls accessibility, not permission to reassign a final field.

Primary note proof

Study-note section OOP07

> final prevents reassignment of a variable, overriding of a method or extension of a class.

Study-note section OOP16

> Fields are hidden rather than dynamically overridden, and same-signature static methods hide rather than override.

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Question sources

Object-Oriented Programming review questions · Question 7

Question 244 | NS0639

Which access modifier allows access ONLY within the same package (not even subclasses in other packages)?

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer: Package-private/default access: omit an access modifier.

OOP08 says an ordinary member with no access modifier grants package access. This admits other code in the same package but does not admit an other-package subclass merely through inheritance. protected would be broader because it adds qualified cross-package subclass access; private would be narrower. The answer is omission, not a default keyword modifier.

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Study-note section OOP08

> Across packages, a subclass accessing a protected instance member must do so through its own inherited context or an appropriate subclass-typed receiver, not any arbitrary superclass object. protected therefore does not mean “subclasses only”: unrelated classes in the same package also have access.

Question sources

Object-Oriented Programming review questions · Question 8

Question 245 | NS0640

Can a constructor be declared private? What is a use case?

Hint

Check the class name, return-type absence and the constructors actually declared.

Answer: Yes. A private constructor can support a factory or controlled singleton creation.

OOP09 explicitly permits private constructors. Restricting constructor access prevents arbitrary outside callers from directly constructing the class, while its own permitted code can still instantiate it. A factory can control creation, or a singleton arrangement can expose a controlled instance. Private construction does not mean the class can never create any object internally.

Primary note proof

Study-note section OOP09

> It initializes the new object and can be overloaded or private. A private constructor can support a factory or controlled singleton creation; it does not mean the class can never create an object internally.

Question sources

Object-Oriented Programming review questions · Question 9

Question 246 | NS0641

What is the difference between a.equals(b) and a == b when comparing objects?

Hint

Separate identity, content equality, immutable values and references that may be reassigned.

Answer: == compares reference identity; equals uses the class's equality policy.

For object references, == asks whether they denote the same object. equals invokes a method whose policy may compare content or value; Object.equals remains identity-based unless overridden. String.equals compares character content. Therefore equals does not automatically mean value equality for every class, and two distinct objects may be equal under an appropriate override.

Primary note proof

Study-note section OOP22

> For object references, == tests identity; equals invokes the class's equality policy. Object.equals uses identity unless overridden; String.equals compares character content. Equal objects must have equal hashCode values when that contract is implemented. toString gives a textual representation, not an equality test.

Question sources

Object-Oriented Programming review questions · Question 10

Question 247 | NS0642

In Java, a class extends two classes simultaneously: public class Dog extends Animal, Mammal {}. What is the result?

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer: Compilation fails: a Java class cannot extend two direct superclasses.

OOP13 explicitly prohibits class Dog extends Animal,Mammal. Java class inheritance permits one direct superclass, so the comma-separated extends clause is invalid rather than a runtime ambiguity to resolve. A class can instead implement multiple interfaces for multiple type contracts, and composition can combine separately held collaborators where multiple state/implementation sources are needed.

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent. A class extends one direct class and can implement multiple interfaces. Single, multilevel and hierarchical inheritance mean one parent, a chain, and several children of one parent. Java does not permit class Dog extends Animal,Mammal.

Study-note section OOP13

> Composition represents has-a, such as Car containing Engine.

Question sources

Object-Oriented Programming review questions · Question 11

Question 248 | NS0643

A private method in a superclass is 'overridden' in a subclass with the same name and signature. What actually happens?

Hint

Check inheritance, signature compatibility, visibility and eligibility for an override.

Answer: The child declaration is independent; it does not override the private parent method.

OOP14 expressly treats a same-signature child method matching a private parent method as an independent declaration. The parent's private method is not an eligible inherited method for runtime overriding. Thus equal spelling and parameters alone do not establish polymorphic replacement, and this is not ordinary static-method hiding either. Calls must follow each declaration's own accessibility and context.

Primary note proof

Study-note section OOP14

> A same-signature child method matching a private parent method is an independent declaration, not runtime overriding and not ordinary static-method hiding.

Study-note section OOP14

> Constructors, final methods, static methods and private parent methods are not overridden.

Question sources

Object-Oriented Programming review questions · Question 12

Question 249 | NS0644

Animal a = new Dog(); - What is the declared type and actual type?

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer: Declared type: Animal. Actual/runtime type: Dog.

Animal is the variable’s declared compile-time type, determining which member operations may be selected by source checking. new Dog() creates the object, making Dog its actual runtime class. Assignment to the Animal reference does not turn that object into an Animal-only instance. Eligible overridden calls still use the runtime Dog implementation.

Primary note proof

Study-note section OOP15

> For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile. If fetch() exists only in Dog, a.fetch() fails compilation. For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Question sources

Object-Oriented Programming review questions · Question 13

Question 250 | NS0645

You have Animal a = new Dog(); and Dog has a method fetch() not in Animal. What happens when you call a.fetch()?

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer: Compilation fails because fetch() is absent from the declared Animal type.

Although the object is a Dog, the reference a has declared type Animal. OOP15 checks member availability using that declared type before considering runtime overriding. Because fetch() exists only in Dog, a.fetch() is rejected. A checked/safe narrowing to a suitable Dog reference could expose Dog-specific members, but the original expression does not perform that narrowing.

Primary note proof

Study-note section OOP15

> For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile. If fetch() exists only in Dog, a.fetch() fails compilation. For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Study-note section OOP21

> Use if(a instanceof Ferrari) before narrowing when the dynamic type is uncertain.

Question sources

Object-Oriented Programming review questions · Question 14

Question 251 | NS0646

What is dynamic binding (late binding) in Java?

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer: Eligible overridden instance methods are selected at runtime from the actual object class.

The compiler first verifies accessible members and selects the method signature from declared types. Runtime dispatch then selects the most specific eligible override in the object's actual class. For Animal a=new Dog(), a.sound() can therefore invoke Dog.sound(). This rule does not make Dog-only methods compile through Animal, and static methods/fields follow different selection rules.

Primary note proof

Study-note section OOP15

> For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile. If fetch() exists only in Dog, a.fetch() fails compilation. For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Study-note section OOP15

> To solve dispatch questions: check the declaration and visibility, determine the compile-time signature, then identify the object's runtime class and its most specific override.

Study-note section OOP16

> Fields are hidden rather than dynamically overridden, and same-signature static methods hide rather than override. With Animal a=new Dog(), a.staticMethod() resolves using Animal's compile-time type. Prefer Animal.staticMethod() for clarity.

Question sources

Object-Oriented Programming review questions · Question 15

Question 252 | NS0647

An abstract class has 3 abstract methods. A subclass implements only 2. What is the result?

Hint

Check whether a class is concrete and whether every inherited abstract contract has an implementation.

Answer: The subclass must remain abstract unless it provides or inherits the third implementation; a concrete incomplete subclass fails compilation.

Implementing only two of three required methods leaves an abstract obligation. OOP18 requires a concrete subclass to provide or inherit implementations for every required method. If the third is still unimplemented, declaring the subclass abstract is permitted; declaring it concrete is not. Merely implementing a majority of the methods does not satisfy the parent contract.

Primary note proof

Study-note section OOP18

> A concrete subclass must provide or inherit implementations of all required abstract methods; otherwise it must itself be abstract.

Question sources

Object-Oriented Programming review questions · Question 16

Question 253 | NS0648

Can an abstract class have a constructor? Can it be instantiated?

Hint

Check whether a class is concrete and whether every inherited abstract contract has an implementation.

Answer: Yes, it can have constructors; no, it cannot be directly instantiated.

OOP18 allows an abstract class to define constructors alongside fields and concrete/abstract methods. Its constructor participates in initializing the inherited part of a concrete subclass object. That initialization role does not permit new of the abstract class itself. Direct instantiation remains prohibited even if the abstract class happens to contain no abstract methods.

Primary note proof

Study-note section OOP18

> An abstract class cannot be directly instantiated but can define fields, constructors, abstract methods and concrete methods. It need not contain an abstract method.

Study-note section OOP10

> Under Java 17 rules, an explicit this(args) or super(args) invocation must be the first constructor statement. this invokes another constructor in the same class; super invokes the immediate superclass constructor. A constructor cannot explicitly invoke both.

Question sources

Object-Oriented Programming review questions · Question 17

Question 254 | NS0649

What is the difference between implementing an interface and extending a class?

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer: extends forms a single superclass relationship; implements fulfills one or more interface contracts.

A class can extend one superclass, gaining a subtype relationship and appropriate inherited implementation/state. It can also implement multiple interfaces, satisfying their required public operations through provided or inherited methods. Interfaces supply capability contracts rather than per-instance fields or constructors. Modern default/static interface methods mean the distinction cannot simply be summarized as interfaces never contain bodies.

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent. A class extends one direct class and can implement multiple interfaces.

Study-note section OOP19

> A class can extend a superclass and implement zero or more interfaces. An interface can extend several interfaces. No interface instance fields or constructors exist; fields are implicitly public static final.

Study-note section OOP19

> A method declaration without private, default or static is implicitly public abstract. Modern interfaces may also have default/static method bodies, and Java 9+ private helper methods. Not every interface method must lack a body in modern Java. Implementations must satisfy all inherited abstract contracts; they may inherit suitable public implementations.

Question sources

Object-Oriented Programming review questions · Question 18

Question 255 | NS0650

For an ordinary bodyless Java interface method declaration with no private, default or static modifier, which two method modifiers are implicit?

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer: public and abstract.

The specified bodyless interface method is implicitly public and abstract. An implementing concrete class must satisfy the public contract without reducing its visibility. This scoped rule does not describe every modern interface method: default and static methods may have bodies, and Java 9 and later allow private helper methods. Those forms are explicitly outside the declaration being classified.

Primary note proof

Study-note section OOP19

> A method declaration without private, default or static is implicitly public abstract. Modern interfaces may also have default/static method bodies, and Java 9+ private helper methods.

Question sources

Object-Oriented Programming review questions · Question 19

Question 256 | NS0651

Can an interface have instance fields (instance variables)?

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer: No. Interface fields are implicitly public static final, not per-instance variables.

OOP19 explicitly excludes interface instance fields. Any field declared in an interface is a public static final class-level constant under the note's rule, even when some modifiers are not written. Implementing classes may define their own instance fields, but those belong to the implementing objects rather than being instance state supplied by the interface.

Primary note proof

Study-note section OOP19

> No interface instance fields or constructors exist; fields are implicitly public static final.

Question sources

Object-Oriented Programming review questions · Question 20

Question 257 | NS0652

What is the output? Animal a = new Cat(); a.makeNoise(); - Cat overrides makeNoise() to print 'Meow'

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer: Meow, from Cat.makeNoise(), assuming the inherited method is accessible and eligible for overriding.

The declared Animal type allows the makeNoise call under the stated override relationship. The actual object is a Cat, so runtime dispatch selects Cat's overriding implementation and prints Meow. Assignment to a superclass reference does not suppress subclass behavior. The question does not specify an exact print API, so no additional punctuation or newline is invented.

Primary note proof

Study-note section OOP15

> For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile. If fetch() exists only in Dog, a.fetch() fails compilation. For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Study-note section OOP15

> To solve dispatch questions: check the declaration and visibility, determine the compile-time signature, then identify the object's runtime class and its most specific override.

Question sources

Object-Oriented Programming review questions · Question 21

Question 258 | NS0653

When is downcasting (Dog dog = (Dog) animal;) safe to use?

Hint

Separate compile-time cast compatibility from the object's actual runtime type.

Answer: For a non-null value, when the actual object is Dog or a subtype; test instanceof when uncertain.

An Animal reference may denote a Dog, another subtype or a plain Animal. Downcasting succeeds for an actual Dog-compatible object and otherwise throws ClassCastException. OOP21 recommends instanceof before narrowing when uncertain. Casting null also succeeds and yields null, but using it as an object afterward fails; therefore successful casting and safe dereferencing are distinct questions.

Primary note proof

Study-note section OOP21

> Upcasting a subclass reference to a superclass is implicit and does not change the object. A downcast may pass compile-time type checks yet fail at runtime. Car a=new Car(); Ferrari f=(Ferrari)a; throws ClassCastException before subsequent drive() or brake() calls. It does not turn a plain Car into a Ferrari.

Study-note section OOP21

> Use if(a instanceof Ferrari) before narrowing when the dynamic type is uncertain. null instanceof Ferrari is false; casting null yields null, although dereferencing it fails.

Question sources

Object-Oriented Programming review questions · Question 22

Question 259 | NS0654

What does the final keyword prevent when applied to a class like: final class String {}?

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer: It prevents subclasses from extending that class.

OOP07 says final on a class prohibits extension. Developers therefore cannot declare a subclass of String. This differs from final on a variable, which prevents reassignment, and final on a method, which prevents overriding. A final class is not automatically immutable merely because subclassing is prohibited; String's immutability is an additional property discussed in OOP22.

Primary note proof

Study-note section OOP07

> final prevents reassignment of a variable, overriding of a method or extension of a class. A final reference may still refer to a mutable object.

Study-note section OOP22

> final class String prevents subclassing; final alone does not make arbitrary objects immutable.

Question sources

Object-Oriented Programming review questions · Question 23

Question 260 | NS0655

Can a static method be overridden in Java?

Hint

Distinguish the current instance, parent context, hidden fields and static dispatch.

Answer: No. A same-signature subclass static method hides the parent static method.

OOP16 states that static methods hide rather than override. Selecting a static method through a reference uses the compile-time type, unlike eligible instance-method dispatch based on the object's runtime class. Thus Animal a=new Dog(); a.staticMethod() selects Animal's static declaration. ClassName.method() is the clearer recommended spelling and makes that selection explicit.

Primary note proof

Study-note section OOP16

> Fields are hidden rather than dynamically overridden, and same-signature static methods hide rather than override. With Animal a=new Dog(), a.staticMethod() resolves using Animal's compile-time type. Prefer Animal.staticMethod() for clarity. Java this and super are unavailable as current-instance references in static context.

Question sources

Object-Oriented Programming review questions · Question 24

Question 261 | NS0656

What is the 'is-a' relationship and how does it relate to inheritance?

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer: A subtype is a valid instance of its parent contract: Dog is an Animal.

OOP13 recommends inheritance when a subtype can satisfy its parent's contract, as Dog satisfies Animal. That is the is-a relationship rather than mere code reuse. Car containing Engine is instead has-a composition. Reusing fields or methods alone does not justify declaring a subclass if its behavior cannot honor the parent's promised operations.

Primary note proof

Study-note section OOP13

> Use inheritance when a subtype can satisfy the parent's contract: Dog is an Animal. For shared Pet.name and speak(), put common state/contract in Pet, specialized fetch() in Dog, and appropriate speak() implementations in Dog and Cat. Composition represents has-a, such as Car containing Engine. Reuse alone does not justify an invalid is-a relationship.

Question sources

Object-Oriented Programming review questions · Question 25

Question 262 | NS0657

You call super() in a subclass constructor but the superclass has no no-arg constructor. What happens?

Hint

Trace constructor delegation and initialization in the specified Java version.

Answer: Compilation fails unless an accessible matching no-argument superclass constructor exists.

An explicit super() requests the immediate superclass's no-argument constructor. OOP10 requires a matching accessible parent constructor, so a superclass with only parameterized constructors cannot satisfy that invocation. The subclass must instead call an appropriate super(arguments) or the superclass must provide an accessible no-argument constructor. Java does not synthesize one after another constructor has been declared.

Primary note proof

Study-note section OOP10

> this invokes another constructor in the same class; super invokes the immediate superclass constructor. A constructor cannot explicitly invoke both. If there is no explicit delegation, an implicit super() is inserted; a matching accessible no-argument parent constructor must exist.

Study-note section OOP09

> If a class declares no constructor, Java supplies a default constructor. If Circle(double r) is declared, new Circle() fails unless a no-argument constructor is also declared.

Question sources

Object-Oriented Programming review questions · Question 26

Question 263 | NS0658

What is the purpose of the protected access modifier specifically for inheritance?

Hint

Locate the accessing code, declaring type, package and receiver before applying visibility rules.

Answer: It permits inherited access in subclasses across packages, and also same-package access.

protected supports subclass use of a parent member without making it universally public. OOP08 also grants access to unrelated code in the same package, so it is not subclasses-only. Across packages an instance member must be accessed in the inherited context or through an appropriate subclass-typed receiver, rather than any arbitrary superclass object.

Primary note proof

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Study-note section OOP08

> Across packages, a subclass accessing a protected instance member must do so through its own inherited context or an appropriate subclass-typed receiver, not any arbitrary superclass object. protected therefore does not mean “subclasses only”: unrelated classes in the same package also have access.

Question sources

Object-Oriented Programming review questions · Question 27

Question 264 | NS0659

An interface Drawable extends both Shape and Colorable interfaces. A class Circle implements Drawable. What must Circle implement?

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer: A concrete Circle must satisfy all inherited abstract operations from Drawable, Shape and Colorable.

Drawable can extend both interfaces, combining their contracts. Circle implements Drawable and must therefore provide or inherit suitable public implementations of every remaining abstract operation from the full hierarchy. One compatible implementation can satisfy a shared signature; inherited defaults may supply implementations where applicable. If obligations remain unresolved, Circle must be abstract rather than an incomplete concrete class.

Primary note proof

Study-note section OOP19

> A class can extend a superclass and implement zero or more interfaces. An interface can extend several interfaces.

Study-note section OOP19

> Implementations must satisfy all inherited abstract contracts; they may inherit suitable public implementations.

Study-note section OOP20

> If two compatible interfaces merely declare the same abstract signature, one public implementation can satisfy both. If return types or other contracts conflict, additional declarations may still be impossible.

Study-note section OOP18

> A concrete subclass must provide or inherit implementations of all required abstract methods; otherwise it must itself be abstract.

Question sources

Object-Oriented Programming review questions · Question 28

Question 265 | NS0660

What happens if you declare the same method in both an abstract class and the interface it implements?

Hint

Check the kind of inherited method and whether competing contracts require an explicit resolution.

Answer: Compatible declarations can coexist; implementation obligations depend on abstract, concrete and default status.

OOP20 says a concrete inherited class method takes precedence over an interface default. An abstract class declaration can instead leave an implementation obligation for a concrete subclass. A compatible public implementation may satisfy the interface contract, but incompatible signatures/return types may be impossible. Without declaration details, these conditional rules are the answer rather than one unconditional outcome.

Primary note proof

Study-note section OOP20

> A concrete inherited class method takes precedence over an interface default. An inherited abstract class declaration can require a concrete subclass to implement the operation rather than silently adopting a default.

Study-note section OOP20

> Unrelated interfaces providing conflicting defaults require an explicit resolution; interfaces do not remove every diamond ambiguity. If two compatible interfaces merely declare the same abstract signature, one public implementation can satisfy both. If return types or other contracts conflict, additional declarations may still be impossible. The answer to “same method in abstract class and interface” depends on whether each is abstract or implemented and whether their signatures are compatible.

Question sources

Object-Oriented Programming review questions · Question 29

Question 266 | NS0661

Can you instantiate an interface? If not, how do you use it?

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer: Not directly. Use an interface reference holding an implementing object, anonymous implementation or suitable lambda.

OOP19 prohibits direct interface instantiation while allowing a reference of interface type to hold an implementing object. Calls then use the interface contract with the object's implementation. Anonymous class syntax still creates an implementing class instance, and a compatible functional-interface lambda supplies an implementation; neither means constructing a bare interface with no implementation.

Primary note proof

Study-note section OOP19

> Implementations must satisfy all inherited abstract contracts; they may inherit suitable public implementations. Interfaces cannot be instantiated directly, but references may hold implementing objects, anonymous implementations or suitable functional-interface lambdas.

Question sources

Object-Oriented Programming review questions · Question 30

Question 267 | NS0662

What is the difference between throw and throws in Java?

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer: throw actively raises an exception value; throws declares a method's possible checked-exception obligation.

OOP30 contrasts throw new IOException("message") with throws IOException in a method signature. The first performs an action in control flow, skipping the remaining try body; the second declares an obligation that callers must handle or propagate. A throws clause does not itself throw anything, and throwing a checked exception still requires handling or declaration.

Primary note proof

Study-note section OOP30

> throw new IOException("message") actively throws a value; throws IOException in a signature declares a possible checked exception. A caller must handle or propagate checked obligations. try encloses guarded operations; a matching catch handles the thrown type or a subtype. Execution skips the remainder of the try when a throw occurs.

Question sources

Object-Oriented Programming review questions · Question 31

Question 268 | NS0663

What happens if an exception occurs in a try block and there is no matching catch block?

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer: It propagates outward to a matching enclosing handler; if uncaught, its thread terminates.

Without a matching local catch, the exception is not ignored and execution does not resume with the next ordinary statement in the try. OOP30 sends it outward through enclosing calls/handlers. Relevant finally blocks normally execute during that exit under OOP31's limits. If no handler catches it, the uncaught exception terminates its thread, not necessarily every JVM thread.

Primary note proof

Study-note section OOP30

> Execution skips the remainder of the try when a throw occurs.

Study-note section OOP30

> If no catch matches, the exception propagates outward and an uncaught exception terminates its thread.

Study-note section OOP31

> finally normally executes after try and any selected catch, including when leaving by return or propagating an exception. It is not guaranteed if the process/JVM exits or crashes, or control never leaves the try.

Question sources

Object-Oriented Programming review questions · Question 32

Question 269 | NS0664

Is it mandatory to handle an ArithmeticException?

Hint

Separate source errors from runtime exceptions and determine the hierarchy and checked obligation.

Answer: No catch-or-declare obligation applies: ArithmeticException is unchecked.

OOP29 lists ArithmeticException among unchecked exceptions, so Java does not require a surrounding catch or throws declaration merely because it may occur. A program can still catch it when that provides useful recovery. Unchecked means no mandatory compile-time handling obligation, not harmless behavior: an uncaught occurrence propagates and may terminate its thread.

Primary note proof

Study-note section OOP29

> RuntimeException and Error subclasses are unchecked; other throwable classes are checked. IOException, FileNotFoundException and SQLException are standard checked examples. ArithmeticException, NullPointerException, ClassCastException and array bounds exceptions are unchecked.

Study-note section OOP29

> Checked does not mean the exception occurs at compile time: the compiler checks an obligation to catch or declare it, while an actual throw occurs at runtime.

Study-note section OOP30

> If no catch matches, the exception propagates outward and an uncaught exception terminates its thread.

Question sources

Object-Oriented Programming review questions · Question 33

Question 270 | NS0665

What is the order of execution in: try { ... } catch(E e) { ... } finally { ... }?

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer: Run try, then a matching catch if thrown, then normally finally.

If try completes normally, no catch executes and finally follows. If an exception is thrown, the remaining try body is skipped, a matching catch runs if available, and finally normally executes before continuation or outward propagation. finally also normally runs on return. Process/JVM exit, a crash or control never leaving try prevents an unconditional always claim.

Primary note proof

Study-note section OOP30

> Execution skips the remainder of the try when a throw occurs.

Study-note section OOP31

> finally normally executes after try and any selected catch, including when leaving by return or propagating an exception. It is not guaranteed if the process/JVM exits or crashes, or control never leaves the try.

Question sources

Object-Oriented Programming review questions · Question 34

Question 271 | NS0666

You catch a general Exception before a specific IOException. What is the result?

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer: Compilation fails because the IOException catch after Exception is unreachable.

Exception already catches IOException values because IOException is a subtype. The later subtype handler can never be selected, and OOP30 says this ordering prevents compilation rather than merely causing the general handler to win at runtime. Place the specific IOException catch before Exception if both handlers are needed.

Primary note proof

Study-note section OOP30

> Catch blocks are considered in source order. Place FileNotFoundException before IOException and IOException before Exception; a general catch before a subtype catch makes that later handler unreachable and prevents compilation.

Question sources

Object-Oriented Programming review questions · Question 35

Question 272 | NS0667

What does e.printStackTrace() actually print?

Hint

Track pending returns or exceptions and distinguish normal cleanup from process termination.

Answer: Exception information plus recorded call-stack frames for diagnosis.

OOP31 distinguishes getMessage, toString and printStackTrace. The stack trace reports exception information together with recorded call frames, helping locate the call path associated with the failure. It is more diagnostic context than the message alone and does not repair or handle the error by itself. Exact text depends on the exception and recorded frames.

Primary note proof

Study-note section OOP31

> getMessage returns the message, toString normally includes type and message, and printStackTrace shows exception information and recorded call frames for diagnosis. Tests should cover expected success, caught failures and propagation. Exception handling does not replace validation or correct an algorithm’s logic automatically.

Question sources

Object-Oriented Programming review questions · Question 36

Question 273 | NS0668

Can a finally block change the return value of a try block?

Hint

Track pending returns or exceptions and distinguish normal cleanup from process termination.

Answer: Yes. A return from finally overrides the pending try/catch return; avoid it.

A try return expression is evaluated before finally runs. A return statement inside finally can replace that pending return and can even suppress an exception, which OOP31 warns against. Merely changing a local after a primitive return expression was evaluated does not automatically recompute that earlier result. The explicit finally return is the clear override mechanism.

Primary note proof

Study-note section OOP31

> A return expression is evaluated before finally; a finally return overrides the pending return and can suppress an exception, so avoid that pattern.

Question sources

Object-Oriented Programming review questions · Question 37

Question 274 | NS0669

What is the difference between Error and Exception in Java?

Hint

Separate source errors from runtime exceptions and determine the hierarchy and checked obligation.

Answer: They are Throwable branches: Error usually denotes serious conditions; Exception contains checked and unchecked application exceptions.

OOP29 places Error and Exception below Throwable. Error normally signals serious conditions for which routine recovery is inappropriate and its subclasses are unchecked. Exception includes unchecked RuntimeException subclasses and other checked exceptions requiring catch-or-declare handling. It is technically possible to catch Error, so the distinction must not be stated as errors can never be caught.

Primary note proof

Study-note section OOP29

> Throwable is the root of throwable values, with Exception and Error branches. RuntimeException and Error subclasses are unchecked; other throwable classes are checked.

Study-note section OOP29

> Checked does not mean the exception occurs at compile time: the compiler checks an obligation to catch or declare it, while an actual throw occurs at runtime. A custom exception can extend Exception or RuntimeException. Error normally signals serious conditions for which routine recovery is inappropriate; it is not technically impossible to catch an Error.

Question sources

Object-Oriented Programming review questions · Question 38

Question 275 | NS0670

FileNotFoundException is a subclass of IOException. Which catch block executes: catch(IOException e) or catch(FileNotFoundException e)?

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer: For FileNotFoundException, the first applicable catch wins; put it before IOException.

Both handler types can match a FileNotFoundException, so catch order matters. If FileNotFoundException is first, that specific block executes. If IOException is placed first with FileNotFoundException afterward, the latter is unreachable and compilation fails. If only IOException is present, it can handle the subclass. The prompt omits order, so these cases must be distinguished.

Primary note proof

Study-note section OOP30

> Execution skips the remainder of the try when a throw occurs.

Study-note section OOP30

> Catch blocks are considered in source order. Place FileNotFoundException before IOException and IOException before Exception; a general catch before a subtype catch makes that later handler unreachable and prevents compilation. If no catch matches, the exception propagates outward and an uncaught exception terminates its thread.

Question sources

Object-Oriented Programming review questions · Question 39

Question 276 | NS0671

What does ClassCastException indicate?

Hint

Separate compile-time cast compatibility from the object's actual runtime type.

Answer: An attempted reference cast is incompatible with the actual object type.

A cast can be syntactically and type-checkably permitted between related reference types while still fail at runtime. OOP21's plain Car cannot become a Ferrari merely by casting, so ClassCastException occurs before subsequent method calls. The exception reports an incompatible dynamic type, not object transformation, allocation or a numeric conversion failure.

Primary note proof

Study-note section OOP21

> Upcasting a subclass reference to a superclass is implicit and does not change the object. A downcast may pass compile-time type checks yet fail at runtime. Car a=new Car(); Ferrari f=(Ferrari)a; throws ClassCastException before subsequent drive() or brake() calls. It does not turn a plain Car into a Ferrari.

Study-note section OOP21

> For Object obj="Hello", (String)obj succeeds and (Integer)obj throws ClassCastException.

Question sources

Object-Oriented Programming review questions · Question 40

Question 277 | NS0672

Can you use multiple catch blocks for one try block? Can one catch block catch multiple exceptions?

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer: Yes to both: use ordered catches or a multi-catch such as catch(IOException | SQLException e).

One try can have several catch blocks; source order and specific-before-general reachability still apply. A single multi-catch can also handle alternative exception types. OOP30 prohibits alternatives related by subclassing, so IOException and FileNotFoundException cannot be paired in one union catch. Their hierarchy instead calls for an appropriate common catch or ordered handlers.

Primary note proof

Study-note section OOP30

> Catch blocks are considered in source order. Place FileNotFoundException before IOException and IOException before Exception; a general catch before a subtype catch makes that later handler unreachable and prevents compilation. If no catch matches, the exception propagates outward and an uncaught exception terminates its thread. Multiple catch blocks are allowed; catch(IOException | SQLException e) handles alternatives, which must not be related by subclassing.

Question sources

Object-Oriented Programming review questions · Question 41

Question 278 | NS0673

What is the output? int x = 5; int y = ++x * x--; System.out.println(x + ' ' + y);

Hint

Track the old and updated values separately and distinguish a character operand from a string.

Answer: 73. The expression leaves x=5 and y=36, then adds the space character's value 32.

Java evaluates ++x first, changing 5 to 6 and supplying 6. x-- then supplies 6 and restores x to 5, so y=36. In x + ' ' + y the space is char, not String, and numeric promotion gives 5+32+36=73. Replacing it with a double-quoted space would instead concatenate separated values.

Primary note proof

Study-note section OOP24

> Starting x=5, printing x++ then ++x gives 5 then 7. For int y=++x*x--, Java uses 6*6=36 and leaves x=5. System.out.println(x + ' ' + y) then adds char space's numeric value 32, so it prints 73. To print separated values, use x + " " + y instead.

Study-note section OOP06

> Arithmetic promotes char, byte and short to int in many expressions.

Question sources

Object-Oriented Programming review questions · Question 42

Question 279 | NS0674

Circle and Sphere are separately declared top-level Java classes. Circle declares private double radius; and Sphere extends Circle. What happens if an instance method of Sphere attempts this.radius = 5.0;?

Hint

Separate parameters from arguments and operations that read state from operations that change it.

Answer: Compilation fails for an ordinary separately declared Sphere: the inherited radius field is private.

The direct field access fails compilation. Circle's private field is not accessible from a separately declared top-level subclass, and this only identifies the current object; it does not relax visibility. Sphere must use an accessible operation supplied by Circle if it needs to request a radius change. Declaring the classes separately removes the shared-nest exception from the scenario.

Primary note proof

Study-note section OOP11

> A private field cannot normally be accessed directly by a separately declared subclass, even with this.field; use a permitted parent operation.

Study-note section OOP08

> For ordinary members: private restricts access to the declaring class/nest; no modifier grants package access; protected grants same-package access plus qualified subclass access; public is widest, provided the declaring type itself is accessible.

Study-note section OOP16

> Java this denotes the current object; this.name=name distinguishes a field from a parameter.

Question sources

Object-Oriented Programming review questions · Question 43

Question 280 | NS0675

What is the difference between a concrete class and an abstract class?

Hint

Check whether a class is concrete and whether every inherited abstract contract has an implementation.

Answer: A concrete class can be instantiated and must satisfy abstract obligations; an abstract class cannot be directly instantiated.

An abstract class can hold state, constructors, concrete methods and abstract methods, and need not contain an abstract method. A concrete subclass must provide or inherit implementations of every required abstract operation before it can serve as an instantiable class. Abstract is therefore a class-level instantiation/contract restriction, not a prohibition on implemented behavior.

Primary note proof

Study-note section OOP18

> An abstract class cannot be directly instantiated but can define fields, constructors, abstract methods and concrete methods. It need not contain an abstract method.

Study-note section OOP18

> A concrete subclass must provide or inherit implementations of all required abstract methods; otherwise it must itself be abstract.

Question sources

Object-Oriented Programming review questions · Question 44

Question 281 | NS0676

String is declared as a final class. What does this mean for developers?

Hint

Decide whether a member belongs to a class or an object, then check initialization and final restrictions.

Answer: Developers cannot subclass String; final prohibits extension.

Applying final to String prevents any class from extending String. It does not prevent constructing or referencing String instances, nor does the class modifier alone explain immutability of arbitrary objects. OOP22 treats String's immutable content as a distinct property. Developers should use existing String operations rather than attempting a String subclass.

Primary note proof

Study-note section OOP07

> final prevents reassignment of a variable, overriding of a method or extension of a class. A final reference may still refer to a mutable object.

Study-note section OOP22

> Java String objects are immutable. text += " World" assigns a reference to a new concatenated result and can print Hello World without mutating the original String object. final class String prevents subclassing; final alone does not make arbitrary objects immutable.

Question sources

Object-Oriented Programming review questions · Question 45

Question 282 | NS0677

What is method hiding in Java and how does it differ from overriding?

Hint

Distinguish the current instance, parent context, hidden fields and static dispatch.

Answer: A subclass static method hides a same-signature parent static method; overriding is runtime instance dispatch.

For static hiding, the selected declaration follows the compile-time type used to qualify the call. For eligible overridden instance methods, the object's runtime class determines the implementation. OOP16's Animal a=new Dog() static call therefore resolves to Animal, unlike a virtual instance sound call that could invoke Dog. Class qualification makes static selection clearer.

Primary note proof

Study-note section OOP16

> Fields are hidden rather than dynamically overridden, and same-signature static methods hide rather than override. With Animal a=new Dog(), a.staticMethod() resolves using Animal's compile-time type. Prefer Animal.staticMethod() for clarity.

Study-note section OOP15

> For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile. If fetch() exists only in Dog, a.fetch() fails compilation. For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Question sources

Object-Oriented Programming review questions · Question 46

Question 283 | NS0678

Given: interface A { void m1(); } interface B { void m2(); } class C implements A, B {} - What must C define?

Hint

Identify the interface contract, effective modifiers and permitted implementation forms.

Answer: C must provide public void m1() and public void m2(), or inherit suitable implementations.

Both unmodified interface declarations are implicitly public abstract. A concrete C implementing A and B must satisfy both signatures, not merely one. In the shown class with no suitable superclass implementation, define public void m1() and public void m2(). Package-private implementations would narrow the public contract; otherwise C must be declared abstract instead of concrete.

Primary note proof

Study-note section OOP19

> A method declaration without private, default or static is implicitly public abstract. Modern interfaces may also have default/static method bodies, and Java 9+ private helper methods. Not every interface method must lack a body in modern Java. Implementations must satisfy all inherited abstract contracts; they may inherit suitable public implementations.

Study-note section OOP18

> A concrete subclass must provide or inherit implementations of all required abstract methods; otherwise it must itself be abstract.

Study-note section OOP14

> Visibility cannot become more restrictive, and checked exceptions cannot be broadened incompatibly.

Question sources

Object-Oriented Programming review questions · Question 47

Question 284 | NS0679

What is the significance of the java.lang.Throwable class?

Hint

Separate source errors from runtime exceptions and determine the hierarchy and checked obligation.

Answer: Throwable is the common root of Java's throwable Error and Exception hierarchy.

OOP29 places both Error and Exception under Throwable, establishing the common type for Java throwable values. The distinction between checked and unchecked branches controls catch-or-declare obligations, not whether an exception happens during compilation. Throwable therefore unifies the hierarchy without implying that every subclass is an ordinary recoverable application exception.

Primary note proof

Study-note section OOP29

> Throwable is the root of throwable values, with Exception and Error branches. RuntimeException and Error subclasses are unchecked; other throwable classes are checked.

Study-note section OOP29

> Checked does not mean the exception occurs at compile time: the compiler checks an obligation to catch or declare it, while an actual throw occurs at runtime. A custom exception can extend Exception or RuntimeException. Error normally signals serious conditions for which routine recovery is inappropriate; it is not technically impossible to catch an Error.

Question sources

Object-Oriented Programming review questions · Question 48

Question 285 | NS0680

A method throws a checked exception but neither catches it nor declares throws. What happens?

Hint

Check the thrown type and catch order before tracing propagation or handler output.

Answer: Compilation fails because the checked exception is neither handled nor declared.

OOP30 requires callers and methods to handle or propagate checked obligations. A method that may throw a checked exception but neither catches it nor declares it in throws violates that compile-time requirement. The failure is not that checked exceptions occur during compilation: the compiler is checking a possible future runtime throw.

Primary note proof

Study-note section OOP30

> throw new IOException("message") actively throws a value; throws IOException in a signature declares a possible checked exception. A caller must handle or propagate checked obligations. try encloses guarded operations; a matching catch handles the thrown type or a subtype.

Study-note section OOP29

> Checked does not mean the exception occurs at compile time: the compiler checks an obligation to catch or declare it, while an actual throw occurs at runtime.

Question sources

Object-Oriented Programming review questions · Question 49

Question 286 | NS0681

What does 'extensibility' mean in the context of polymorphism?

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer: New subtypes can supply behavior while existing common-type client code remains reusable.

OOP15 describes a common Animal array loop invoking speak on Dog and Cat objects. Adding another conforming Animal subtype can supply its own override without rewriting that generic loop. That ability is extensibility through polymorphism: clients depend on the shared contract, while runtime dispatch supplies subtype behavior. The new subtype must still satisfy that contract.

Primary note proof

Study-note section OOP15

> To solve dispatch questions: check the declaration and visibility, determine the compile-time signature, then identify the object's runtime class and its most specific override. An array Animal[]{new Dog(),new Cat()} can call speak() on each using one loop. New subclasses can supply behavior without rewriting this generic loop, which is the practical meaning of extensibility.

Question sources

Object-Oriented Programming review questions · Question 50

Question 287 | NS0682

You have: Object obj = 'Hello'; String s = (String) obj; Is this valid? What about Integer i = (Integer) obj;?

Hint

Separate compile-time cast compatibility from the object's actual runtime type.

Answer: Literal single-quoted Hello is invalid. With "Hello", the String cast succeeds and the Integer cast throws ClassCastException.

A Java character literal cannot contain the multi-character text 'Hello', so the printed declaration fails before either cast executes. OOP21 explicitly separates that defect from the intended double-quoted version. After changing it to a String literal, obj refers to a String: narrowing back to String succeeds, but narrowing it to Integer fails at runtime.

Primary note proof

Study-note section OOP21

> For Object obj="Hello", (String)obj succeeds and (Integer)obj throws ClassCastException. A literal 'Hello' is invalid Java syntax; use double quotes for a string literal before tracing the program.

Study-note section OOP-S04

> A double-quoted string literal denotes a String object. new String(existingString) creates a String with the same text; new String(charArray) copies the array's characters into an immutable String. A single-quoted literal denotes one char, so several characters enclosed together in single quotes cannot supply a String.

Question sources

Object-Oriented Programming review questions · Question 51

Question 288 | NS0683

If a subclass constructor doesn't explicitly call super(), what does Java do?

Hint

Trace constructor delegation and initialization in the specified Java version.

Answer: If no explicit this(...) or super(...) delegation exists, Java inserts super(); this(...) instead delegates within the class.

Omitting explicit super() alone is insufficient to determine what happens. A constructor may explicitly call this(...), delegating to another same-class constructor that eventually reaches superclass construction. Only when there is no explicit constructor delegation does OOP10 insert super(), requiring an accessible matching parent no-argument constructor. Otherwise the program fails to compile.

Primary note proof

Study-note section OOP10

> Under Java 17 rules, an explicit this(args) or super(args) invocation must be the first constructor statement. this invokes another constructor in the same class; super invokes the immediate superclass constructor. A constructor cannot explicitly invoke both. If there is no explicit delegation, an implicit super() is inserted; a matching accessible no-argument parent constructor must exist.

Question sources

Object-Oriented Programming review questions · Question 52

Question 289 | NS0734

What is the difference between a class and an object? Use a concrete example.

Hint

Distinguish an object’s stored state, operations, identity and class definition.

Answer: A class defines common structure/behavior; an object is one instance, such as a particular Rectangle with width and height.

Rectangle can define width, height, getArea and getPerimeter for every rectangle. A particular rectangle object stores its own width and height values and responds to those operations. OOP01 distinguishes that instance from the type description: two objects can have equal field values and still have distinct identities. The class is shared structure, not one particular rectangle.

Primary note proof

Study-note section OOP01

> An object has identity, state and behavior. State is stored in fields; behavior is provided by methods. A Rectangle object may hold width and height and answer getArea() or getPerimeter(). Two objects can have equal fields and still be distinct objects. A class defines their common structure and behavior; an object is a particular instance.

Question sources

Object-Oriented Programming review questions · Question 1

Question 290 | NS0735

Why is encapsulation important? How is it achieved?

Hint

Identify whether the emphasis is grouping state, hiding implementation, exposing essentials or varying behavior.

Answer: Encapsulation protects invariants and stable interfaces by grouping state with validated operations and controlling access.

OOP02 groups state and operations with controlled access, keeping representation decisions behind stable interfaces. A private account balance exposed through deposit(amount) can reject invalid negative changes rather than permitting unrestricted assignment. This localizes responsibility and lets clients use meaningful operations without depending on internal representation. OOP11 cautions that automatically exposing setters for every field is not required.

Primary note proof

Study-note section OOP02

> Encapsulation groups related state and operations and controls access. Information hiding keeps implementation decisions behind a stable interface. Abstraction presents the essential operations while omitting details irrelevant to the caller. These ideas overlap: a private balance plus deposit(amount) can demonstrate all three, depending on what the question emphasizes.

Study-note section OOP11

> An accessor/getter returns information; a mutator/setter changes state. Encapsulation does not require blindly generating setters for every field: validate changes and expose only useful operations. For a bank account, deposit must reject negative amounts.

Question sources

Object-Oriented Programming review questions · Question 2

Question 291 | NS0736

What is the difference between method overloading and method overriding? Provide examples.

Hint

Compare names and parameter signatures before considering return types or access modifiers.

Answer: Overloading changes parameter signatures; overriding replaces eligible inherited instance behavior with the same signature.

For overloading, f(int), f(double) and f(int,int) share a name but have distinct signatures selected using declared arguments at compile time. For overriding, Dog extends Animal and supplies void sound() matching Animal's eligible void sound(); a superclass reference can then dispatch to Dog at runtime. Changing only the return type cannot create an overload, and override return compatibility still matters.

Primary note proof

Study-note section OOP12

> Overloaded methods share a name but have different parameter types, counts or type order. Return type alone is insufficient in Java and C++. Overloads can also arise through inherited methods; "same class only" is a teaching simplification. f(int), f(double) and f(int,int) are distinct, but changing a parameter name does not create an overload.

Study-note section OOP12

> The compiler selects the applicable signature from the declared argument types and conversions.

Study-note section OOP14

> A subclass overrides an eligible inherited instance method with the same signature and a compatible return type; a covariant reference return may be a subtype. Visibility cannot become more restrictive, and checked exceptions cannot be broadened incompatibly.

Study-note section OOP15

> For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Question sources

Object-Oriented Programming review questions · Question 3

Question 292 | NS0737

Why can't Java classes inherit from multiple parent classes? How is this limitation overcome?

Hint

Identify the subtype or ownership relationship and distinguish class inheritance from interface contracts.

Answer: Java permits one direct superclass. Use multiple interfaces for type contracts and composition for cooperating implementation/state.

The immediate reason the declaration is rejected is Java's one-direct-superclass rule in OOP13. Multiple interfaces supply multiple type contracts, while contained collaborators model has-a reuse without multiple class inheritance. This does not eliminate every diamond problem: OOP20 requires explicit resolution of conflicting unrelated interface defaults. The note supplies these mechanisms, rather than a historical language-design rationale.

Primary note proof

Study-note section OOP13

> class Dog extends Animal declares a subclass/derived/child of a superclass/base/parent. A class extends one direct class and can implement multiple interfaces. Single, multilevel and hierarchical inheritance mean one parent, a chain, and several children of one parent. Java does not permit class Dog extends Animal,Mammal.

Study-note section OOP13

> Composition represents has-a, such as Car containing Engine. Reuse alone does not justify an invalid is-a relationship.

Study-note section OOP20

> Unrelated interfaces providing conflicting defaults require an explicit resolution; interfaces do not remove every diamond ambiguity.

Question sources

Object-Oriented Programming review questions · Question 4

Question 293 | NS0738

What is polymorphism? Explain with a code example.

Hint

Check what compiles from the declared type, then select behavior from the actual runtime class.

Answer: Polymorphism lets a common-type reference invoke behavior appropriate to the actual subtype.

Example: class Animal { void sound() { System.out.println("Animal"); } } class Dog extends Animal { void sound() { System.out.println("Bark"); } } Then Animal a=new Dog(); a.sound(); prints Bark. The declared Animal type makes sound available; the actual Dog object supplies its eligible override at runtime. A new Animal subtype can participate through the same common operation.

Primary note proof

Study-note section OOP15

> For Animal a=new Dog(), Animal is the declared type and Dog the actual type. The declared type determines which member operations compile. If fetch() exists only in Dog, a.fetch() fails compilation. For an eligible overridden sound(), a.sound() invokes Dog.sound() at runtime.

Study-note section OOP15

> To solve dispatch questions: check the declaration and visibility, determine the compile-time signature, then identify the object's runtime class and its most specific override. An array Animal[]{new Dog(),new Cat()} can call speak() on each using one loop. New subclasses can supply behavior without rewriting this generic loop, which is the practical meaning of extensibility.

Question sources

Object-Oriented Programming review questions · Question 5

Question 294 | NS0739

When would you use an abstract class versus an interface?

Hint

Check the kind of inherited method and whether competing contracts require an explicit resolution.

Answer: Use an abstract class for shared state/construction/implementation; use an interface for capabilities across unrelated classes and multiple type contracts.

An abstract class can centralize common fields, constructor initialization and concrete operations while leaving selected operations abstract. An interface specifies a capability that otherwise unrelated classes can implement, and a class may implement several interfaces. Modern interface defaults do not supply per-instance fields or constructors, so shared object state/construction remains a reason to choose an abstract class.

Primary note proof

Study-note section OOP20

> Use an abstract class for shared state, construction and common implementation; use an interface for a capability across unrelated classes and multiple type inheritance.

Study-note section OOP19

> No interface instance fields or constructors exist; fields are implicitly public static final.

Study-note section OOP18

> An abstract class cannot be directly instantiated but can define fields, constructors, abstract methods and concrete methods.

Question sources

Object-Oriented Programming review questions · Question 6