CS Exit Ethiopia

Complete Study Notes

Computer Science Exit Examination

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 6

How to use these study notes

START01 A practical study cycle

Start with a small set of questions before reading.

Mark the exact thing that stopped you: an unknown term, a rule, a calculation, a trace or a choice between alternatives.

Use the question lookup to open the relevant section.

Read one section, close it, and explain the rule in your own words.

Solve the question again and say why each other choice fails.

Then change one assumption or input and solve the variation.

This checks understanding beyond recognition.

A useful 60-minute session: 10 minutes of cold questions, 20 minutes of focused reading, 20 minutes of closed-book solving and 10 minutes updating an error log.

Adjust the timing to your needs.

Revisit mistakes after a gap, for example the next day, several days later and the following week.

These intervals are a practical starting plan, not a fixed learning law.

Record the section ID, your mistaken rule, the corrected rule and one boundary case.

Keep the log short enough to review.

Mix related courses during revision: programming with data structures; databases with security; networking with administration.

Explain how the same word changes meaning across contexts.

START02 What to emphasize

Emphasize definitions, key differences, algorithm costs, SDLC models, databases, networking and security.

Use short oral questions to check understanding and correct common misconceptions.

Combine those habits with the tracing, calculation, design and diagnosis demanded by the Ministry learning outcomes.

Allocate revision by both exam share and your error log.

A course with fewer items can still deserve more time if your foundations are weak.

The blueprint is a planning baseline, not a prediction of exact questions.

Definitions: distinguish nearby concepts, not just their names.

Calculations: write units, formulas, assumptions and intermediate steps.

Code and SQL: check validity before tracing the text; state any changes before analyzing a corrected version.

Algorithms and protocols: know the input, decision rule, state change, termination and failure cases.

Design choices: state the criterion.

A choice can change when cost, reliability, consistency or latency matters more.

Course group	Items in the 100-item blueprint	Revision implication
Web Programming	9	Reserve extra practice for HTML, CSS, JavaScript, PHP, client/server behavior and safe data handling.
Data Structures and Algorithms	7	Trace operations and compare their assumptions and costs.
Each of the other fourteen courses	6 each	Study all sixteen; do not abandon a whole course based on a short note or an alleged repeated-question list.

START03 How to handle exam traps

Underline NOT, FALSE, BEST and EXCEPT before comparing choices.

Check whether the stem supplies enough facts to select one answer.

A long scenario is not automatically an analysis question, and a familiar answer letter is not evidence.

When code is invalid, a diagram is absent, or two choices express the same rule, identify the clarification needed before answering.

Learn the correct rule and the smallest clarification needed.

Do not memorize an arbitrary key as a universal fact.

Distinguish worst case from average case, and an upper bound from a tight bound.

Check language version, SQL dialect, addressing assumptions, graph weights and tie-breaking rules.

For subnetting and scheduling, calculate rather than guess from a familiar number.

Use timed mixed papers only after you can explain your errors.

Revisit the reading sections behind recurring mistakes.

Object Oriented Programming

Complete Study Notes

OOP01 | Objects, classes and responsibilities

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.

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.

Object Oriented Programming module

OOP02 | Four principles and the difference between hiding and abstraction

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.

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.

Object Oriented Programming module

OOP03 | Create objects and distinguish references from instances

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.

Assignment t=s copies a reference, so both can refer to the same object; changing that object through t is observable through s.

Java passes all arguments by value.

Passing a reference copies it: a method may modify the shared object, but assigning its parameter to a new object does not reassign the caller's reference.

The runtime may reclaim unreachable objects; collection timing is unspecified.

Reachable object cycles can matter for retention, but an unreachable cycle can be collected.

Object Oriented Programming module

OOP04 | UML class and instance notation

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.

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.

Object Oriented Programming module

OOP05 | Variables, defaults, scope and initialization

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.

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.

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

A final local is permitted; final is not an access modifier.

Object Oriented Programming module

OOP06 | Primitive types, literals and conversions

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

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.

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.

Object Oriented Programming module

OOP07 | Static members, static blocks and final

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.

A static initializer runs during class initialization, before the class’s first relevant active use, not simply whenever a file is loaded. 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.

A calculation using only parameters may be static.

Object Oriented Programming module

OOP08 | Access control and package boundaries

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.

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.

Object Oriented Programming module

JLS 21: protected access

OOP09 | Constructors and missing default constructors

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.

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.

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.

Object Oriented Programming module

OOP10 | Constructor chaining and initialization order

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.

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.

Java 25 permits restricted statements before explicit constructor invocation; the older first-statement rule must be version qualified.

Object Oriented Programming module

Java 25: flexible constructor bodies

OOP11 | Methods, getters and setters

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.

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.

A private field cannot normally be accessed directly by a separately declared subclass, even with this.field; use a permitted parent operation.

Object Oriented Programming module

OOP12 | Method overloading and compile-time selection

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.

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.

Distinguish a different signature from overriding an inherited signature.

Object Oriented Programming module

OOP13 | Inheritance and useful relationships

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.

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.

Object Oriented Programming module

OOP14 | Overriding and restrictions

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.

A same-signature child method matching a private parent method is an independent declaration, not runtime overriding and not ordinary static-method hiding.

An abstract inherited method is implemented by an overriding method in a concrete subclass.

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

Object Oriented Programming module

OOP15 | Dynamic dispatch: declared type versus actual type

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.

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.

Object Oriented Programming module

OOP16 | this, super, fields and static hiding

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.

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.

Object Oriented Programming module

OOP17 | C++ this and language-specific assumptions

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.

C++ runtime polymorphism requires a virtual function when calling through a base pointer/reference.

C++ may inherit multiple classes; Java permits multiple interface types instead.

C++ object-value assignment can copy an object, whereas assigning Java class variables copies references.

Never transfer Java dispatch or memory-management assumptions to a C++ question without checking the code.

C++ draft: this pointer

Object Oriented Programming module

OOP18 | Abstract classes and concrete implementations

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.

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.

Object Oriented Programming module

OOP19 | Interfaces: contracts and modern Java

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.

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.

Interfaces cannot be instantiated directly, but references may hold implementing objects, anonymous implementations or suitable functional-interface lambdas.

Object Oriented Programming module

JLS 21: interface methods and conflicts

OOP20 | Conflicting interface methods and abstract class choice

Use an abstract class for shared state, construction and common implementation; use an interface for a capability across unrelated classes and multiple type inheritance.

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.

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.

Object Oriented Programming module

JLS 21: interface methods and conflicts

OOP21 | Casts and instanceof

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.

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.

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.

Object Oriented Programming module

OOP22 | Identity, equals, toString and immutable strings

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.

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. 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.

Object Oriented Programming module

OOP23 | Expressions: numbers, bits and text output

Java evaluates operands left to right while respecting precedence.

With n=6,m=15,p=3, n-=m then n*=p leaves -27. 6-2+10%4+7 is 13.

In 8-2*5*(3+9/3)+6/3, the result is -50.

Integer 6/4 is 1; double 6.0/4 is 1.5.

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

For bitwise values, write binary columns: 11&9=9; 9^3=10; 10|12=14. && and || short circuit Boolean operands; & and | can also operate on Boolean values without short circuiting.

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

Object Oriented Programming module

OOP24 | Increment order and the character-space trap

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.

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

When x and y are numeric, x + ' ' + y performs numeric addition; x + " " + y concatenates a separating space.

Use the expression's operand types to determine the result.

JLS 21: expression order and string addition

OOP25 | Control flow traces and exact printing

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

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".

A loop summing j=1..10 yields 55.

In if(a)... else if(a&&b)..., the second condition can never be true on a path reaching it because a is already false.

Object Oriented Programming module

OOP26 | Arrays: creation, bounds and ragged rows

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.

Elements have a declared common component type, which may be a superclass/interface.

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.

Object Oriented Programming module

OOP27 | Array algorithms, sort and copy APIs

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.

java.util.Arrays.sort(a) sorts in place.

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.

Random access is the reliable array advantage; size is bounded by implementation/resources, not unlimited.

Object Oriented Programming module

OOP28 | Packages, imports and command-line tools

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. java.lang is implicitly imported; java.util supplies utilities; java.io supplies I/O; java.awt includes Button and Checkbox.

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.

For ordinary public top-level types, filename and type name match.

Capitalized class names and lowercase method names are conventions, not universal syntax requirements.

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

Object Oriented Programming module

OOP29 | Exception hierarchy and checked obligations

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.

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.

Object Oriented Programming module

JLS 21: checked and unchecked exceptions

OOP30 | throw, throws and catch selection

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.

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.

Object Oriented Programming module

JLS 21: checked and unchecked exceptions

OOP31 | finally, cleanup and diagnostic information

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.

A return expression is evaluated before finally; a finally return overrides the pending return and can suppress an exception, so avoid that pattern.

try-with-resources closes AutoCloseable resources even on exceptional exit. 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.

Object Oriented Programming module

JLS 21: try and finally

OOP32 | Streams, GUI events and JDBC practice

InputStream/OutputStream read and write bytes; Reader/Writer handle text characters through an encoding.

Close resources deterministically.

A read returning -1 signals end-of-stream.

A GUI event source notifies registered listeners; a button can call an ActionListener’s actionPerformed method.

In Swing, update components on the event dispatch thread and keep handlers short so the interface remains responsive.

Object Oriented Programming module

Oracle: event dispatch thread

OOP33 | JDBC flow from user action to database

A JDBC operation obtains a Connection, prepares SQL, binds values, executes and processes results, then closes resources.

For PreparedStatement ps=con.prepareStatement("SELECT name FROM Student WHERE id=?"), ps.setInt(1,id) binds the first parameter. executeQuery returns a ResultSet; rs.next() moves to each row before reading columns. executeUpdate returns an affected-row count for updates.

Parameter binding keeps values separate from SQL syntax.

Catch or declare SQLException.

When several writes form one operation, disable auto-commit, commit on success and roll back on failure.

A GUI handler should delegate a slow database operation away from its UI thread, then display the result through the appropriate UI mechanism.

Oracle: prepared JDBC statements

OOP34 | Legacy applets and lifecycle questions

In the historical applet model, Java code was embedded in an HTML document and executed by a compatible JVM host/plugin or applet viewer. 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 applets as historical Java concepts rather than modern browser deployment tools. 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.

A JVM alone is also not the complete browser embedding environment.

Oracle: applet lifecycle callbacks

OOP35 | Android Activity, Intent, manifest and layouts

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.

Android: Intents and extras

Android: Activities

OOP36 | Android app declarations and packaging

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.

Traditional view-based layout containers such as LinearLayout and RelativeLayout derive from android.view.ViewGroup, which contains child views.

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.

A manifest describes the package, while layout resources describe views.

Use these distinctions to compare Android application structure with Java language concepts.

Android: manifest

Android: ViewGroup

Android: packages

OOP37 | Validate Java syntax before tracing

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.

For a valid Java loop with i=2,4,6,8,10 that prints i*i only when i%4==0, the printed values are 16 and 64.

A valid implementation can define public String welcome() and additionally overload welcome(int).

Identify an implementation by its complete declarations and method bodies.

Check the required signature before selecting an implementation.

A rounded mathematical mean is not necessarily the exact Java printed output.

Object Oriented Programming module

OOP38 | Apply Java rules under explicit assumptions

Several Java statements can be valid at once: a static constant and a parameter-only utility may both be suitable static members; multiple array declarations/initializers can be legal; an abstract class may contain a complete interface implementation.

Encapsulation and abstraction overlap, so choose only when the requested emphasis is specific.

An interface supports multiple type inheritance and abstraction. “All methods are public abstract” applies only under a restricted abstract-declaration or legacy interpretation. Java protected also grants access within the same package, so it cannot enforce “only subclasses.”

No keyword implements overriding: it is recognized from compatible declarations, and @Override is an annotation.

For C++ this, ordinary non-static member functions have an implicit object; variables do not receive a hidden argument.

Object Oriented Programming module

OOP39 | Language classifications and portability

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.

Portability still depends on compatible APIs, versions and environment assumptions; it is not an unconditional guarantee for any program on any device.

Java supports objects and classes but also primitive values. “Difficult to learn” is a subjective judgment rather than a technical language property.

Syntax conventions, bytecode portability and runtime implementation must be distinguished.

Object Oriented Programming module

University of Rhode Island: conventional language generation taxonomy

OOP-S01 | Identifiers, literals and source comments

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.

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.

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.

Object Oriented Programming module · page 4

Java SE 17 JLS - Identifiers

OOP-S02 | Java 17 floating-point evaluation and printed doubles

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.

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.

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.

Java SE 17 JLS - Floating-point expressions and multiplication

Java SE 17 - Double.toString(double)

Java SE 17 - PrintStream.print(double)

OOP-S03 | Expression results, assignment and control headers

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.

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.

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.

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.

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.

For Boolean values, ! reverses true and false, && is true only when both operands are true, and || is true when at least one is true. Short-circuit evaluation skips an unnecessary right operand. For each binary column, integral & produces 1 only for two 1 bits, | produces 1 for either 1 bit, and ^ produces 1 for differing bits.

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.

Java SE 17 JLS - Operators and expressions

Java SE 17 JLS - Statements and loop headers

Object Oriented Programming module · page 7

Oracle Java tutorial · Boolean and conditional operators

Oracle Java tutorial · Bitwise operators

Oracle Java tutorial · Branching statements

OOP-S04 | Core package membership and text values

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.

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.

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.

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.

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.

Java SE 17 - Exception

Java SE 17 - java.util package

Java SE 17 - java.net package

Java SE 17 - String

Oracle Java 17 Character API

OOP-S05 | Inheritance boundaries and whole-part relationships

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.

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.

Object Oriented Programming module · page 16

Object Oriented Programming module · page 15

OMG UML 2.5.1 - Property aggregation semantics

OOP-S06 | Introductory Java characteristics

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.

Object Oriented Programming module · page 2