CS Exit Ethiopia

# Revision Outlines

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 3

# Object Oriented Programming

Revision Outlines

## Objects, classes and responsibilities | Exam focus

- Object versus class
  - Identity, state, behavior
  - Fields store state
  - Methods provide behavior
  - Rectangle: width, height
  - getArea(), getPerimeter()
  - Equal fields, distinct identities
  - Class: common structure and behavior
  - Object: particular instance
- Responsibility-driven classes
  - Account controls balance
  - Transaction records payment
  - Service coordinates objects
  - Messages: method calls, obj.add()
  - Delegation to other objects
  - Objects manage own state
  - Procedural and OO reuse
  - Procedural global shared data optional

## Encapsulation, hiding and abstraction | Exam focus

- Encapsulation, hiding and abstraction
  - Encapsulation: state/operations grouped, access controlled
  - Information hiding: implementation behind stable interface
  - Abstraction: essential operations, irrelevant details omitted
  - private balance plus deposit(amount)
  - Overlapping principles, question emphasis matters
- Inheritance and polymorphism
  - Inheritance: subtype relationship, possible implementation reuse

- Common type, concrete-specific behavior
- Fields/methods wrapping: encapsulation
- What versus how: abstraction
- Precise mechanism over vague slogan

## References, instances and value passing | Exam focus

- Reference declaration versus object creation
  - Student s;: reference only
  - new Student(): allocate and initialize
  - s.name, s.study(): visibility-dependent access
  - t=s: reference copy
  - Shared object aliasing
  - Mutation through t visible through s
- Java value passing and reachability
  - All arguments passed by value
  - Copied reference permits shared-object mutation
  - Parameter reassignment leaves caller reference unchanged
  - Unreachable objects reclaimable
  - Collection timing unspecified
  - Reachable cycles may retain objects
  - Unreachable cycles collectible

## UML class and instance notation | Exam focus

- UML class box
  - Name, attributes, operations
  - + public, - private, # protected
  - name:String
  - getName():String
  - Student(String,String,String,int)
  - Three strings plus integer
  - Age "25": wrong argument type
- UML instance notation
  - Underlined instanceName:ClassName

- sara:Student
- Class notation: type
- Instance notation: particular object
- Optional slot values
- Static-feature underlining: different meaning
- Whole notation determines interpretation

## Variables, defaults and initialization

• Local, instance and static variables
- Locals: methods, constructors, blocks
- Definite assignment before use
- Locals: no public/private/protected
- Fields: class, outside method bodies
- Instance fields: per object
- Static fields: per class
- Location alone doesn't distinguish field kinds

• Java defaults and initialization
- Fields and array elements defaulted
- Numeric zero, char '\u0000'
- boolean false, references null
- Unassigned local int t: compilation error
- int a=10,b=10: initial values
- final local permitted
- final isn't access modifier

## Primitive types and conversions

• Java primitive and reference types
- byte, short, int, long
- Signed widths: 8,16,32,64 bits
- byte: -128..127
- short: -32768..32767
- float, double, char, boolean
- char: unsigned 16-bit UTF-16 unit

- Digit, letter, symbol, not whole String
- enum and String: reference types

• Conversions and arithmetic promotion
- Compatible widening often implicit
- Narrowing usually explicit, possibly lossy
- int→float widening may lose precision
- Widening-losslessness claim false
- char/byte/short often promoted to int
- boolean ↔ numeric conversion forbidden
- if(0), numeric-zero loop condition: invalid

## Static members and final

• Static class members
- One field per loaded class
- Shared across instances
- Three constructor increments: count=3
- Prefer ClassName.count, ClassName.utility()
- Static method: no current instance
- Explicit object permits instance-member access

• Static initialization and final
- Class initialization before relevant active use
- Not merely file loading
- final variable: no reassignment
- final method: no overriding
- final class: no extension
- final reference may target mutable object
- PI: static final candidate
- Student name: normally instance state
- Parameter-only calculation: possible static method

## Access control and packages

• Java access boundaries
- private: declaring class/nest

- No modifier: package access
- protected: package plus qualified subclass access
- public: widest ordinary access
- Declaring type must be accessible
- Top-level class/interface: public or package-private
- Top-level private/protected forbidden
- Member types: additional visibility possibilities

• Protected access across packages
- Own inherited context allowed
- Appropriate subclass-typed receiver allowed
- Arbitrary superclass receiver disallowed
- Same-package unrelated classes also allowed
- protected ≠ subclasses only
- Own main accesses own instance's private fields

## Constructors and generated defaults | Exam focus

• Constructor identity and purpose
- Exact class name
- No return type, including void
- Initialize new object
- Overloaded/private constructors permitted
- Private: factory or controlled singleton
- Internal object creation still possible

• Default-constructor conditions
- No declared constructor → compiler default
- Circle(double r) suppresses generated no-arg
- new Circle(): explicit no-arg then required
- Constructors neither inherited nor overridden
- return; allowed, return expression forbidden
- Field defaults precede ordinary constructor initialization
- Defaults independent of generated constructor

## Constructor chains and initialization order | Exam focus

- Java17 constructor delegation
  - this(args)/super(args): first statement
  - this: same-class constructor
  - super: immediate parent constructor
  - Cannot explicitly invoke both
  - Absent explicit delegation: implicit super()
  - Accessible matching no-arg parent required
- Construction order and version conditions
  - Parent construction first
  - Subclass field initializers/instance blocks in order
  - Relevant subclass constructor body last
  - FinalExam parent: num1=4
  - Subclass initializer: num=3
  - this(5): num=25
  - No-arg body: num1=num=25
  - Java25: restricted pre-invocation statements
  - First-statement rule requires version qualification

## Methods, accessors and mutators | Exam focus

- Method structure and invocation
  - Modifiers, return type, name
  - Parameter list and body
  - public int getAge(){return age;}
  - void: no returned value
  - Parameters: declared inputs
  - Arguments: call-supplied values
  - object.method(): instance call
  - Implicit this in suitable instance context
- Accessors, mutators and validation
  - Getter: return information
  - Setter: change state
  - Useful operations over blanket setters

- Validate state changes
- deposit: reject negative amounts
- Separate subclass: private parent field inaccessible
- this.field doesn't bypass access
- Use permitted parent operation

## Overloads and compile-time selection | Exam focus

- Method overload distinctions
  - Same name, different types/count/type order
  - Return type alone insufficient
  - Java and C++ rule
  - Inherited methods can form overloads
  - Same-class-only claim simplified
  - f(int), f(double), f(int,int)
  - Parameter-name change: no overload
- Compile-time overload selection
  - Declared argument types and conversions
  - Exact int preferred over double widening
  - Return type outside ordinary Java signature
  - Different signature versus inherited-signature override

## Inheritance and composition | Exam focus

- Class inheritance forms
  - Dog extends Animal
  - Subclass/derived/child, superclass/base/parent
  - One direct superclass
  - Multiple implemented interfaces
  - Single: one parent
  - Multilevel: inheritance chain
  - Hierarchical: siblings share parent
  - extends Animal,Mammal: invalid Java
- Contract-based inheritance versus composition
  - Subtype satisfies parent contract

- Dog is an Animal
- Pet: common name and speak contract
- Dog: specialized fetch
- Dog/Cat: appropriate speak implementations
- Composition: has-a
- Car contains Engine
- Reuse alone doesn't justify is-a

## Overriding rules | Exam focus

- Override eligibility and restrictions
  - Inherited instance method, same signature
  - Compatible return type
  - Covariant reference return subtype
  - Visibility cannot narrow
  - Checked exceptions cannot broaden incompatibly
  - @Override: compiler-checking annotation
  - No Java override keyword
  - Constructors/final/static/private methods: no overriding
- Independent declaration versus override
  - Private-parent same signature: independent child declaration
  - No runtime overriding
  - No ordinary static-method hiding
  - Concrete subclass implements inherited abstract method
  - Unrelated same-name methods: no overriding

## Declared types and dynamic dispatch | Exam focus

- Declared type versus runtime class
  - Animal a=new Dog()
  - Animal: declared, Dog: actual
  - Declared type controls compilable operations
  - Dog-only fetch: a.fetch() fails
  - Eligible sound override: Dog.sound() runtime
- Dispatch-solving sequence

- Declaration and visibility first
- Compile-time signature second
- Runtime class and most-specific override third
- Animal[]{Dog,Cat}: common speak loop
- New subclasses supply behavior
- Generic loop unchanged: extensibility

## this, super and hiding

- this, super and field selection
  - this: current object
  - this.name=name: field versus parameter
  - super.name: hidden parent field
  - super.sound(): parent implementation
  - No second parent object
  - Unqualified instance call may dispatch dynamically
  - Explicit super call differs
- Field and static-method hiding
  - Fields aren't dynamically overridden
  - Same-signature static methods hide
  - Animal a=new Dog(): static resolves Animal
  - Prefer Animal.staticMethod()
  - Static context: no current this/super

## C++ this and object rules

- C++17 this pointer
  - Ordinary non-static member function
  - Pointer to called object
  - this->field: member selection
  - Implicit object argument explanation
  - Member variables don't receive argument
  - Static member function: no this
- C++ versus Java object rules
  - Base-pointer/reference dispatch requires virtual

- C++: multiple class inheritance
- Java: multiple interface types
- C++ object-value assignment may copy object
- Java class-variable assignment copies reference
- Check dispatch and memory-management assumptions

## Abstract classes and implementations | Exam focus

• Abstract class capabilities
- No direct instantiation
- Fields, constructors, concrete methods allowed
- Abstract methods allowed, not required
- Abstract method: no body
- abstract plus final: conflicting requirements
- Required versus forbidden overriding

• Concrete implementation obligations
- Provide/inherit all required abstract implementations
- Otherwise subclass must remain abstract
- Required: String welcome()
- int welcome(): incompatible return
- welcome(int): overload only
- Valid implementation plus unrelated overload allowed

## Interfaces and modern Java | Exam focus

• Interface contract and implementation
- Payable: double getPaymentAmount()
- Implicit public abstract operation
- Invoice implements Payable
- Required public double getPaymentAmount()
- getQuantity()*getPricePerItem()
- Class: superclass plus zero/multiple interfaces
- Interface extends multiple interfaces
- No instance fields or constructors
- Fields: public static final

- Modern interface method rules
  - No private/default/static: public abstract
  - default/static methods may have bodies
  - Java9+: private helpers
  - All-interface-methods-bodyless claim outdated
  - All inherited abstract contracts satisfied
  - Suitable public implementation may be inherited
  - No direct interface instantiation
  - References hold implementing objects
  - Anonymous implementations
  - Suitable functional-interface lambdas

## Defaults, conflicts and class choice

- Abstract class versus interface choice
  - Abstract class: shared state/construction/implementation
  - Interface: unrelated-class capability
  - Multiple type inheritance
  - Concrete class method beats interface default
  - Inherited abstract class declaration may prevail
  - Concrete subclass may need explicit implementation
- Interface conflicts and diamond ambiguity
  - Unrelated conflicting defaults: explicit resolution
  - Interfaces don't eliminate all diamond ambiguity
  - Same compatible abstract signature: one public implementation
  - Conflicting return types/contracts may be impossible
  - Abstract versus implemented status matters
  - Signature compatibility determines answer

## Casts and instanceof

- Upcasting versus downcasting
  - Implicit upcast leaves object unchanged
  - Downcast: compile-time acceptance, runtime risk
  - Car a=new Car()

- (Ferrari)a: ClassCastException
- Later drive()/brake() calls unexecuted
- Cast doesn't transform object

• Safe narrowing and null
- instanceof before uncertain downcast
- null instanceof Ferrari: false
- Casting null yields null
- Dereferencing null fails
- Object obj="Hello": (String)obj succeeds
- (Integer)obj: ClassCastException
- 'Hello': invalid Java literal
- Clarify single-quoted bank version first

## Identity, equality and immutable strings

• Identity, equality and representation
- Reference ==: identity
- equals: class equality policy
- Object.equals: identity unless overridden
- String.equals: character content
- Equal objects require equal hashCode
- toString: textual representation, not equality

• Immutable String and text APIs
- text += " World": new concatenated result
- Hello World without original String mutation
- final String class: no subclassing
- final alone doesn't ensure arbitrary immutability
- charAt(index): char
- length(): UTF-16 units
- substring(begin,end): exclusive end
- Character.toUpperCase(ch): character conversion
- matches(String regex): regular-expression parameter

## Numeric, bitwise and Boolean expressions

- Java expression tracing
  - Left-to-right operand evaluation with precedence
  - n=6,m=15,p=3: n-=m,n*=p → -27
  - 6-2+10%4+7=13
  - 8-2*5*(3+9/3)+6/3=-50
  - Integer 6/4=1
  - Double 6.0/4=1.5
  - Java % accepts floating operands
  - C++ % differs
- Bitwise and Boolean operations
  - Binary-column method
  - 11&9=9
  - 9^3=10
  - 10|12=14
  - &&, ||: Boolean short circuit
  - Boolean &, |: no short circuit
  - Printed true: lowercase
  - Neither 1 nor TRUE

## Increment order and concatenation traps

- Increment and character-space trap
  - x=5: x++ then ++x → 5,7
  - y=++x*x--: 6*6=36
  - Final x=5
  - x + ' ' + y: arithmetic
  - Space char value=32
  - println result: 73
  - x + " " + y: separation
- String concatenation grouping
  - Left-to-right + groups
  - String operand starts concatenation
  - 1+2+"x" → "3x"

- "x"+1+2 → "x12"
- Single-quoted space doesn't separate values
- String concatenation and evaluation order

## Control-flow traces and exact output

• Control flow and exact printing
- while/for: pre-test, possibly zero
- do-while: post-test, at least once
- break: exit loop
- continue: remaining body skipped
- for continue then update
- Colon switch: fall-through without break
- Choice 1: Hello then you
- break then outside Goodbye
- default skipped

• Execution counts versus printing
- i=0..4: conditional even-i print
- Unconditional print(i+1)
- Exact output: "0 122 344 5"
- Sum j=1..10: 55
- if(a) then else if(a&&b)
- Second condition unreachable as true
- Reached else implies a=false

## Array creation, bounds and ragged rows

• Array declarations and component types
- int[] a=new int[5]: valid
- int a[]={1,2,3}: valid
- int[] a={}: zero-length
- Declaration initializer needn't use new
- int a[],b: only a array
- int[] c,d: both arrays
- Object length fixed after creation

- Reference may be reassigned
- Common declared component type
- Superclass/interface component type permitted

• Bounds and ragged arrays
- Valid indices: 0..a.length-1
- Invalid access: ArrayIndexOutOfBoundsException
- Two-dimensional: arrays of arrays
- Ragged {{12,-9,8},{7,14},{-32,-1,0}}
- Row lengths: 3,2,3
- No portable physical-contiguity guarantee
- No raw-address assumptions

## Array algorithms, sorting and copying

• Array sum and average
- Four elements: a[i]=i*3
- Sum: 0+3+6+9=18
- Average: sum every element
- Floating division by length
- Zero length handled separately
- Representation/printing versus rounded result
- Bank mean: 98.8/6≈16.46666667
- Not exact printed-string assertion

• Array sorting, copying and aliasing
- java.util.Arrays.sort(a): in-place
- System.arraycopy(src,srcPos,dst,dstPos,length)
- Existing destination array
- arraycopy: lowercase c
- Arrays.copyOf(a,n): new copy
- a=b: same-array alias
- Random access: reliable advantage
- Size bounded by resources/implementation

## Packages, imports and Java tools | Exam focus

- Packages and import order
  - package p; before imports
  - Imports before top-level declarations
  - import p.*: accessible p types
  - No automatic p.sub imports
  - Name organization and access control
  - Subpackages gain no special access
  - java.lang: implicit import
  - java.util: utilities
  - java.io: I/O
  - java.awt: Button, Checkbox
- Compilation, launch and identifiers
  - javac Test.java → bytecode
  - java Test → class launch
  - Modern java Test.java: source-file launch
  - Unrestricted run question: multiple answers possible
  - Public top-level type: matching filename
  - Class/method capitalization: conventions
  - Identifiers case sensitive
  - $ allowed, @ forbidden

## Exception hierarchy and checked obligations | Exam focus

- Throwable hierarchy and source errors
  - Throwable → Exception, Error
  - RuntimeException/Error subclasses: unchecked
  - Other throwable classes: checked
  - IOException, FileNotFoundException, SQLException: checked
  - ArithmeticException, NullPointerException: unchecked
  - ClassCastException, array bounds exceptions: unchecked
  - Missing semicolon/undeclared variable: source errors
  - Source errors aren't checked exceptions
- Checked obligations versus runtime throwing

- Compiler checks catch-or-declare obligation
- Actual exception thrown at runtime
- Custom extends Exception or RuntimeException
- Error: normally serious conditions
- Routine recovery usually inappropriate
- Catching Error technically possible

## throw, throws and catch selection | Exam focus

- throw versus throws
  - throw new IOException("message"): active throw
  - throws IOException: signature declaration
  - Caller catches or propagates checked obligations
  - try: guarded operations
  - catch: matching thrown type/subtype
  - Throw skips remaining try statements
- Catch selection and propagation
  - Source-order handler selection
  - FileNotFoundException before IOException
  - IOException before Exception
  - General-first subtype catch: unreachable, compilation failure
  - No match: outward propagation
  - Uncaught exception terminates its thread
  - Multiple catch blocks allowed
  - catch(IOException | SQLException e)
  - Multi-catch alternatives cannot subclass each other

## finally, cleanup and diagnostics

- finally and pending control transfer
  - Normally after try/selected catch
  - Return and propagation included
  - JVM/process exit or crash: exception
  - Never leaving try: exception
  - Return expression evaluated before finally

- finally return overrides pending return
- Potential exception suppression
- Avoid returning from finally

• Resource cleanup and diagnostics

- try-with-resources: closes AutoCloseable
- Exceptional exit cleanup
- getMessage: message
- toString: normally type and message
- printStackTrace: exception and recorded call frames
- Test success, caught failure, propagation
- Handling doesn't replace validation
- No automatic logic correction

## Streams and GUI events | Exam focus

• Streams and GUI events

- InputStream/OutputStream: bytes
- Reader/Writer: text characters through encoding
- Deterministic resource closure
- read()=-1: end-of-stream
- Event source notifies registered listeners
- Button → ActionListener.actionPerformed
- Swing updates: event dispatch thread
- Short handlers preserve responsiveness

## JDBC queries and transactions | Exam focus

• JDBC query lifecycle

- Connection → prepare SQL → bind values
- Execute → process results → close
- PreparedStatement: SELECT name FROM Student WHERE id=?
- ps.setInt(1,id): first parameter
- executeQuery → ResultSet
- rs.next() before column reads
- executeUpdate → affected-row count

- SQL binding, transactions and responsiveness
  - Bound values separate from SQL syntax
  - Catch/declare SQLException
  - Multiple writes: disable auto-commit
  - Success: commit
  - Failure: rollback
  - Slow database work off GUI thread
  - Display through appropriate UI mechanism

## Historical applet lifecycle

- Historical applet environment and lifecycle
  - Java embedded in HTML
  - Compatible JVM host/plugin or applet viewer
  - init: initialize once
  - start: after initialization, resume
  - stop: host deactivation, suspend
  - destroy: final cleanup before unloading
  - paint: drawing, not constructor
- Applet historical interpretation limits
  - Historical bank concepts
  - Not modern browser deployment instructions
  - destroy not exclusive to browser shutdown
  - Host may destroy on unloading
  - Abrupt termination: cleanup unguaranteed
  - JVM alone insufficient for browser embedding

## Android Activity and Intent

- Android interaction and communication components
  - Activity: user-interaction entry point
  - Often corresponds to screen
  - Intent: requested action/explicit target
  - Extras carry inter-activity values
  - putExtra("id",17): destination reads ID

- Content provider: structured-data sharing
- Broadcast receiver: broadcast handling
- Neither substitutes parameterized activity launch

## Android declarations and packaging

- Manifest and layout containers
  - AndroidManifest.xml: essential app metadata
  - Components, permissions, intent filters
  - Manifest isn't executable code
  - Not all source/layout details
  - LinearLayout, RelativeLayout derive ViewGroup
  - android.view.ViewGroup contains child views
- Android packaging distinctions
  - .apk: Android package archive
  - Installation and runtime contents
  - Android Package Kit: common exam expansion
  - Documentation: Android package
  - Manifest describes package
  - Layout resources describe views
  - Old Android questions outside Java module

## Malformed syntax and trace corrections

- OCR repair before literal tracing
  - extends misread as Extends
  - println misread as printIn/printin
  - Constructor names, braces, <=, == corrupted
  - Case-sensitive Java may fail compilation
  - Intended trace requires stated repair
  - Even i=2,4,6,8,10; i%4==0
  - Only 16 and 64 print
  - Extra offered values incorrect
- SS118 fragments and numeric output
  - Recovered public String welcome() implementation

- Additional welcome(int) overload
- Current A-D fragments unreliable
- Required signature over fabricated letter
- Rounded mean choice ≠ exact printed output

## Ambiguous bank interpretations

- Overlapping single-choice options
  - Static constant and parameter-only utility suitable
  - Multiple valid array declarations/initializers
  - Abstract class may implement complete interface
  - Encapsulation/abstraction overlap
  - Specific requested emphasis needed
- Interpretation-dependent bank claims
  - Interface: multiple type inheritance plus abstraction
  - Broad purpose choices overlap
  - All-public-abstract: restricted/legacy interpretation
  - protected includes same-package access
  - No overriding keyword
  - Compatible declarations establish overriding
  - @Override: annotation
  - C++ hidden object: non-static member functions
  - Not arguments passed to member variables

## Language classifications and portability

- Language taxonomy and portability
  - C++, Java, Python, Perl: third generation
  - Taxonomy separate from object orientation
  - Taxonomy separate from compilation/interpretation
  - Java source → JVM bytecode
  - Compatible JVM plus libraries: cross-system portability
- Portability qualifications and language properties
  - Compatible APIs, versions, environment assumptions
  - No unconditional any-device guarantee

- Java objects/classes plus primitives
- Learning difficulty: subjective judgment
- Syntax conventions versus bytecode portability
- Runtime implementation distinguished