Module-6 Core Java

1. Introduction to Java

I. History of Java

1. Origins: Designed by James Gosling at Sun Microsystems (started 1991, public 1995).
2. Goal: "Write once, run anywhere" — portable bytecode that runs on any JVM.
3. Evolution: Major milestones — Java 1.0 (1995), introduction of generics (Java 5), lambdas (Java 8), module system (Java 9), and ongoing LTS releases.

II. Features of Java

1) simple

2) OO

3) interpreter : JVM : bytecode to machine code

4) robust

5) secure

6) dynamic

7) high performance : 10x

8) multithreading

9) platform independent

10) portable

III. Understanding JVM, JRE, and JDK

1. JDK (Java Development Kit): Tools to develop — javac, jar, plus JRE.
2. JRE (Java Runtime Environment): JVM + standard libraries to run programs.
3. JVM (Java Virtual Machine): Executes bytecode; provides class loading, memory management, and runtime.

IV. Setting up Java environment and IDE

1. Install JDK appropriate for your OS.
2. Set JAVA_HOME environment variable and add JAVA_HOME/bin to PATH.

3. Choose IDE: IntelliJ (feature-rich), Eclipse (popular in academia), or VS Code (lightweight).
4. Create a new Java project, configure SDK/JDK, and add/run your first HelloWorld class.

V. Java Program Structure

1. Package declaration (optional): package com.example;.
2. Imports for external classes: import java.util.*;.
3. Class definition with fields and methods.
4. main method signature for entry point: public static void main(String[] args).

2. Data Types, Variables, and Operators

I. Primitive Data Types

1. Integer types: byte (8-bit), short (16-bit), int (32-bit), long (64-bit).
2. Floating point: float (32-bit), double (64-bit).
3. Character: char (16-bit Unicode).
4. Boolean: boolean (true / false).

II. Variable Declaration and Initialization

1. Declaration: int x;
2. Initialization: int x = 5;
3. Local vs instance vs static variables: scope and lifetime differ.

III. Operators

1. Arithmetic: + - * / % — watch integer division vs float.
2. Relational: ==, !=, >, <, >=, <= — return boolean.
3. Logical: &&, ||, ! — short-circuit behavior for &&/||.
4. Assignment: =, +=, -=, *=, /= etc.
5. Unary: ++, --, +, - (prefix vs postfix semantics).
6. Bitwise: &, |, ^, ~, <<, >>, >>> — operate on binary form.

IV. Type Conversion and Type Casting

1. Widening (implicit): int → long → float → double.
2. Narrowing (explicit): cast required, e.g. int i = (int) 3.9;.
3. Be careful with overflow/precision loss when narrowing.

3. Control Flow Statements

I. If-Else Statements

1. Syntax: if(condition) { } else if(cond2) { } else { }.
2. Use braces for multi-line blocks to avoid bugs.

II. Switch Case Statements

1. Syntax uses case labels and break (or fall-through deliberately).
2. Since Java 14+: switch expressions and -> syntax available.

III. Loops (For, While, Do-While)

1. for(init; condition; update) { } — preferred when count known.
2. while(condition) { } — condition-first loop.
3. do { } while(condition); — executes body at least once.

IV. Break and Continue Keywords

1. break: exits the nearest loop or switch.
2. continue: skips remaining loop body and proceeds to next iteration.
3. Labeled break/continue exist for nested loops: break outer;.

4. Classes and Objects

I. Defining a Class and Object in Java

1. Class syntax: class ClassName { fields; methods; }.
2. Object creation: ClassName obj = new ClassName();.

II. Constructors and Overloading

1. Constructor: special method with no return, name equals class.
2. Overloading: multiple constructors with different parameters.

III. Object Creation and Accessing Members

1. Use new to instantiate.
2. Access fields/methods via obj.field or obj.method() (consider access modifiers).

IV. this Keyword

1. this refers to the current instance.
2. Use to resolve parameter-shadowed fields: this.name = name;.

5. Methods in Java

I. Defining Methods

1. Signature: [modifiers] returnType name(params) { body }.
2. Example: public int add(int a, int b) { return a + b; }.

II. Method Parameters and Return Types

1. Parameters are passed by value — for objects, the reference is passed by value.
2. Use void for no return; otherwise return the declared type.

III. Method Overloading

1. Same method name, different parameter types or counts.
2. Compiler resolves call at compile-time (static binding).

IV. Static Methods and Variables

1. static members belong to the class, not instances.
2. Access via ClassName.method(); cannot use instance this inside static methods.

6. Object-Oriented Programming (OOPs) Concepts

I. Basics of OOP

1. Encapsulation: hide internal state with private and provide getters/setters.
2. Inheritance: extend classes to reuse behavior.
3. Polymorphism: same interface, multiple implementations.
4. Abstraction: expose only necessary details (abstract classes / interfaces).

II. Inheritance Types

1. Single: one parent.
2. Multilevel: chain of inheritance (A -> B -> C).
3. Hierarchical: one parent, many children.

III. Method Overriding and Dynamic Dispatch

1. Overriding: subclass provides new implementation for inherited method (same signature).
2. Dynamic method dispatch: runtime decides which override to call based on object type.

7. Constructors and Destructors

I. Constructor Types

1. Default (no-arg) constructor — provided if none declared.
2. Parameterized constructors — accept arguments for initialization.

II. Copy Constructor (Emulated)

1. Java has no built-in copy constructor; implement one: Class(Class other) { this.field = other.field; }.

III. Constructor Overloading

1. Provide multiple constructors to support different initialization needs.

IV. Object Life Cycle and Garbage Collection

1. Objects created with new live until unreachable.
2. JVM GC reclaims memory — do not rely on finalize() (deprecated).

3. Use try-with-resources or explicit close for external resources (I/O).

8. Arrays and Strings

I. One-Dimensional and Multidimensional Arrays

1. 1D: int[] a = new int[5]; or int[] a = {1,2,3};.
2. 2D: int[][] m = new int[3][4]; — access m[i][j].

II. String Handling: String, StringBuffer, StringBuilder

1. String immutable — operations create new objects.
2. StringBuffer (synchronized) and StringBuilder (unsynchronized) for mutable sequences — use for loops/concatenation.

III. Array of Objects

1. Example: Person[] arr = new Person[3]; arr[0] = new Person("A"); — array holds references.

IV. String Methods

1. length(), charAt(index), substring(start,end), indexOf(), toUpperCase(), trim(), split().

9. Inheritance and Polymorphism

I. Inheritance Types and Benefits

1. Reuse code, extend behavior, model is-a relationships.

II. Method Overriding

1. Subclass implements method with same signature — @Override annotation recommended.

III. Dynamic Binding (Run-Time Polymorphism)

1. Reference of base type pointing to subclass object: Animal a = new Dog(); a.speak(); — Dog's method called.

IV. Super Keyword and Method Hiding

1. super() calls parent constructor.
2. super.method() invokes parent class method.

3. Method hiding: static methods are hidden, not overridden — binding is compile-time.

10. Interfaces and Abstract Classes

I. Abstract Classes and Methods

1. Abstract class: may contain both implemented and abstract methods.
2. Subclass must implement abstract methods or be abstract itself.

II. Interfaces: Multiple Inheritance in Java

1. Before Java 8: interfaces only abstract methods.
2. Java 8+: default and static methods allowed.
3. A class can implement multiple interfaces — solves multiple inheritance of behavior.

III. Implementing Multiple Interfaces

1. Provide implementations for all abstract methods across interfaces.
2. Resolve diamond conflicts by overriding and specifying which default is chosen.

11. Packages and Access Modifiers

I. Java Packages

1. Built-in packages: java.lang, java.util, java.io, etc.
2. User-defined: organize classes by namespace, e.g. package com.example.app;.

II. Access Modifiers

1. private: class-only.
2. default (package-private): accessible within same package.
3. protected: package + subclasses.
4. public: accessible from anywhere.

III. Importing Packages and Classpath

1. import brings classes into scope.
2. Classpath controls where JVM/compiler looks for classes — set via -cp or project settings.

12. Exception Handling

I. Types of Exceptions

1. Checked exceptions: must be declared or handled (e.g., IOException).
2. Unchecked exceptions: RuntimeException and its subclasses (e.g., NullPointerException).

II. try, catch, finally, throw, throws

1. try { } catch(Exception e) { } finally { } — finally executes regardless.
2. throw new Exception() to throw; throws on method signature to declare checked exceptions.

III. Custom Exception Classes

1. Create by extending Exception (checked) or RuntimeException (unchecked).
2. Provide constructors to pass messages or causes.

13. Multithreading

I. Introduction to Threads

1. A thread is a lightweight path of execution within a process.
2. Java supports multiple threads to perform concurrent tasks.

II. Creating Threads

1. Extend Thread and override run() or implement Runnable and pass to Thread.
2. Use ExecutorService for thread pooling and better management.

III. Thread Life Cycle

1. States: New, Runnable, Running, Blocked/Waiting, Timed Waiting, Terminated.
2. Methods: start(), run(), join(), interrupt().

IV. Synchronization and Inter-thread Communication

1. synchronized blocks/methods to protect shared data (mutual exclusion).
2. wait(), notify(), notifyAll() for coordination (must be used within synchronized context).
3. Prefer higher-level concurrency utilities from java.util.concurrent (locks, semaphores, concurrent collections).

14. File Handling

I. Introduction to File I/O

1. java.io and java.nio packages provide file I/O.
2. Streams (byte) vs Readers/Writers (character).

II. FileReader and FileWriter

1. FileReader reads characters; FileWriter writes characters.
2. Use with buffering for performance.

III. BufferedReader and BufferedWriter

1. Wrap FileReader/FileWriter in BufferedReader/BufferedWriter for efficient line-based I/O.
2. Use readLine() to read text line-by-line.

IV. Serialization and Deserialization

1. Implement Serializable to serialize objects to streams.
2. Use ObjectOutputStream and ObjectInputStream.
3. Be mindful of serialVersionUID and transient fields.

15. Collections Framework

I. Introduction

1. High-level data structures and algorithms (List, Set, Map, Queue).
2. Prefer interfaces (List, Set, Map) in signatures and specific implementations when constructing.

II. List, Set, Map, Queue Interfaces

1. List: ordered collection (allows duplicates).
2. Set: unordered collection of unique elements.
3. Map: key-value pairs.
4. Queue: FIFO data structures (used for scheduling).

III. Common Implementations

1. ArrayList (resizable array), LinkedList (doubly-linked),
2. HashSet (hash-based unique set), TreeSet (sorted set),
3. HashMap (hash-based map), TreeMap (sorted map by keys).

IV. Iterators and ListIterators

1. Iterator for forward traversal and safe removal (iterator.remove()).
2. ListIterator for bidirectional traversal and modification.

16. Java Input/Output (I/O)

I. Streams in Java

1. Byte streams: InputStream / OutputStream.
2. Character streams: Reader / Writer.

II. Reading and Writing Data Using Streams

1. For binary data: FileInputStream/FileOutputStream.
2. For text: FileReader/FileWriter wrapped with BufferedReader/BufferedWriter.

III. Handling File I/O Operations

1. Use try-with-resources (try (resource) { }) to auto-close streams.
2. Handle IO exceptions (IOException) with try/catch or throws.
3. For large files or high-performance needs, use NIO (Files, Paths, FileChannel).