Module 6 – Core Java (Theory)

1. Introduction to Java

i. History of Java

Java was developed by James Gosling and colleagues at Sun Microsystems and first released in 1995. It was created to provide a platform-independent, object-oriented language for consumer devices and evolved into a general-purpose language used for server-side, desktop, mobile, and embedded applications.

ii. Features of Java (Platform Independent, Object-Oriented, etc.)

- Simple
- Object Oriented
- Interpreter
- Robust
- Secure
- Dynamic
- High performance
- Multithreading
- Platform independent
- Portable

iii. Understanding JVM, JRE, and JDK

- **JVM (Java Virtual Machine)*:*** Runtime engine that executes Java bytecode and provides platform abstraction, garbage collection, and security checks.
- **JRE (Java Runtime Environment)*:*** Contains the JVM and standard class libraries required to run Java applications.
- **JDK (Java Development Kit)*:*** Complete kit for Java development — includes the JRE plus developer tools such as javac (compiler), javadoc, jar, and debugging tools.

iv. Setting up the Java environment and IDE (Eclipse, IntelliJ)

- Install JDK and set JAVA_HOME.
- Add java and javac to PATH.
- Choose an IDE such as Eclipse or IntelliJ IDEA and configure the project SDK to the installed JDK.

v. Java Program Structure (Packages, Classes, Methods)

- Program is organized into *packages*, *classes*, and *methods*.
- A typical file contains a package declaration, import statements, one public class matching the filename, fields, constructors, and methods (including `public static void main(String[] args)` as the entry point).

2. Data Types, Variables, and Operators

i. Primitive Data Types in Java (int, float, char, etc.)

byte, short, int, long (integers); float, double (floating-point); char (single 16-bit Unicode character); boolean (true/false).

ii. Variable Declaration and Initialization

Declaration: int a;

Initialization: int a = 5;

iii. Operators: Arithmetic, Relational, Logical, Assignment, Unary, and Bitwise

- **Relational**: ==, !=, >, <, >=, <=.
- **Logical**: &&, ||, !.
- **Assignment**: =, +=, -=, *=, /=, %=.
- **Unary**: ++, --, +, -, !.
- **Bitwise**: &, |, ^, ~, <<, >>, >>>.

iv. Type Conversion and Type Casting

- Implicit widening: smaller to larger type (e.g., int → long).
- Explicit narrowing (casting): int i = (int) 3.14; may lose data.

  Automatic promotion occurs in expressions; be mindful of integer division vs. floating point.

3. Control Flow Statements

i. If-Else Statements

If the condition is **true**, the statements inside the **if block** are executed; otherwise, the statements inside the **else block** are executed.

ii. Switch Case Statements

The switch statement allows a variable to be tested for equality against multiple values.

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

- **For Loop:**
  Used when the number of iterations is known.

- **While Loop:**
  Used when the number of iterations is not known in advance.

- **Do-While Loop:**
  Executes the loop body at least once before checking the condition.

iv. Break and Continue Keywords

- **Break Statement:**
  Used to terminate the loop or switch statement immediately.

- **Continue Statement:**
  Used to skip the current iteration and move to the next one.

4. Classes and Objects

i. Defining a Class and Object in Java

A class is a blueprint (fields + methods). An object is an instance of a class.

ii. Constructors and Overloading

A constructor initializes an object. Overloading allows multiple constructors with different parameter lists.

iii. Object Creation, Accessing Members of the Class

Use new to create objects: ClassName obj = new ClassName(); Access members via obj.field or obj.method().

iv. this Keyword

Refers to the current object; used to disambiguate instance variables from parameters and to call other constructors (this(...)).

5. Methods in Java

i. Defining Methods

Methods have signature: [modifiers] returnType name(parameterList) { body }.

ii. Method Parameters and Return Types

Methods can accept parameters and return values; use void for no return.

iii. Method Overloading

Same method name with different parameter types/counts; resolved at compile time.

iv. Static Methods and Variables

static members belong to the class, not instances. Call static methods using ClassName.method().

6. Object-Oriented Programming (OOPs) Concepts

i. Basics of OOP: Encapsulation, Inheritance, Polymorphism, Abstraction

- **Encapsulation** — Hiding internal state using private fields and providing access via getters/setters.
- **Inheritance** — Mechanism to derive a new class from an existing class (single inheritance for classes). Promotes code reuse.
- **Polymorphism** — Ability of objects to take multiple forms; includes compile-time (overloading) and runtime polymorphism (method overriding).
- **Abstraction** — Exposing only necessary features via abstract classes or interfaces; hides implementation details.

ii. Inheritance: Single, Multilevel, Hierarchical

- Single Inheritance - In single inheritance, a class inherits from only one parent class.
- Multilevel Inheritance - In multilevel inheritance, a class is derived from another class, which is also derived from another class.
- Hierarchical Inheritance - In hierarchical inheritance, multiple classes inherit from a single parent class.

iii. Method Overriding and Dynamic Method Dispatch

Subclass can override superclass methods; actual method chosen at runtime based on object type.

7. Constructors and Destructors

i. Constructor Types (Default, Parameterized)

Default (no-arg provided by compiler if none exists) and Parameterized constructors.

ii. Copy Constructor (Emulated in Java)

Java does not have a built-in copy constructor but you can implement one:

public ClassName(ClassName other) { this.field = other.field; ... }

iii. Constructor Overloading

Multiple constructors with different parameters.

iv. Object Life Cycle and Garbage Collection

JVM automatically reclaims memory of objects no longer referenced. Finalization (finalize()) exists historically but is deprecated; rely on try-with-resources and explicit resource management.

8. Arrays and Strings

i. One-Dimensional and Multidimensional Arrays

- One-dimensional: int[] arr = new int[5];
- Multidimensional: int[][] mat = new int[3][4];
- Array of objects: MyClass[] objs = new MyClass[n];

ii. String Handling in Java: String Class, StringBuffer, StringBuilder

- String is immutable. Methods: length(), charAt(), substring(), indexOf(), equals(), compareTo(), toLowerCase(), toUpperCase().
- StringBuffer and StringBuilder are mutable; use StringBuilder for single-threaded performance.

iii. Array of Objects

In Java, an array of objects is used to store multiple objects of the same class.

iv. String Methods (length, charAt, substring, etc.)

length(), charAt(), substring(), indexOf(), equals(), compareTo(), toLowerCase(), toUpperCase().

9. Inheritance and Polymorphism

i. Inheritance Types and Benefits

Types of inheritance — Single, Multi-level, Hierarchical, Mutiple, Hybrid.

Benefits of Inheritance — Code reuse, logical hierarchy, extensibility.

ii. Method Overriding

Subclass provides specific implementation; use @Override annotation.

iii. Dynamic Binding (Run-Time Polymorphism)

Method calls on supertype references invoke subclass implementations at runtime.

iv. Super Keyword and Method Hiding

super accesses superclass members; static method hiding occurs when subclass defines a static method with same signature — hiding is resolved at compile time.

10. Interfaces and Abstract Classes

i. Abstract Classes and Methods

Can have abstract methods (no body) and concrete methods. Use when classes share behavior but cannot be instantiated.

ii. Interfaces: Multiple Inheritance in Java

Define method contracts. From Java 8 onward interfaces can have default and static methods; Java 9+ allows private methods. A class can implement multiple interfaces (provides multiple inheritance of type).

iii. Implementing Multiple Interfaces

Use implements and provide implementations for all abstract methods.

11. Packages and Access Modifiers

i. Java Packages: Built-in and User-Defined Packages

Organize classes into namespaces. Use package declaration. Java has built-in packages (e.g., java.util) and user-defined packages.

ii. Access Modifiers: Private, Default, Protected, Public

- private: accessible only within class.
- default (package-private): accessible within package.
- protected: accessible within package and subclasses.
- public: accessible from everywhere.

iii. Importing Packages and Classpath

Use import to access classes from other packages. Classpath tells JVM where to find classes; set via -cp or environment variables.

12. Exception Handling

i. Types of Exceptions: Checked and Unchecked

Checked (must be declared/handled, e.g., IOException) and Unchecked (runtime exceptions like NullPointerException).

ii. try, catch, finally, throw, throws

Use finally for cleanup; try-with-resources is preferred for AutoCloseable resources.

iii. Custom Exception Classes

Extend Exception or RuntimeException to create domain-specific exceptions.

13. Multithreading

i. Introduction to Threads

Thread is a path of execution. Useful for concurrent tasks.

ii. Creating Threads by Extending Thread Class or Implementing Runnable Interface

Two approaches: extend Thread class or implement Runnable (or use Callable and ExecutorService for tasks returning results).

iii. Thread Life Cycle

New → Runnable → Running → Blocked/Waiting → Terminated.

iv. Synchronization and Inter-thread Communication

Use synchronized blocks/methods, wait(), notify(), notifyAll(), and higher-level concurrency utilities from java.util.concurrent (locks, semaphores, executors) to avoid race conditions.

14. File Handling

i. Introduction to File I/O in Java (java.io package)

Java I/O lives in java.io and java.nio packages. Use FileReader, FileWriter, BufferedReader, BufferedWriter for character streams and FileInputStream, FileOutputStream for byte streams.

ii. FileReader and FileWriter Classes

iii. BufferedReader and BufferedWriter

Improve performance by reducing I/O calls.

iv. Serialization and Deserialization

Convert objects to bytes via Serializable interface and use ObjectOutputStream / ObjectInputStream to persist and restore object state.

15. Collections Framework

i. Introduction to Collections Framework

The **Collections Framework** in Java is a unified architecture for representing and manipulating groups of objects. It provides interfaces, implementations (classes), and algorithms to store, retrieve, and process data efficiently.

ii. List, Set, Map, and Queue Interfaces

- **List Interface**
  - Maintains an **ordered collection** of elements.
  - Allows **duplicate elements**.
- **Set Interface**
  - Represents a **collection of unique elements** (no duplicates allowed).
- **Map Interface**
  - Stores elements as **key-value pairs**.
- **Queue Interface**
  - Used to store elements in a **FIFO (First-In-First-Out)** manner.

iii. ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap

- **ArrayList**
  - Implements the List interface.
  - Stores elements in a **dynamic array**.
- **LinkedList**
  - Implements both List and Queue interfaces.
- **HashSet**
  - Implements the Set interface.
  - Uses a **hash table** for storage.
- **TreeSet**
  - Implements the NavigableSet interface.
  - Stores elements in a **sorted (ascending) order**.
- **HashMap**
  - Implements the Map interface.
  - Stores **key-value pairs** using a **hash table**.
- **TreeMap**
  - Implements the NavigableMap interface.

iv. Iterators and ListIterators

Use Iterator and ListIterator to traverse collections safely. For concurrency, use concurrent collections in java.util.concurrent.

16. Java Input/Output (I/O)

i. Streams in Java (InputStream, OutputStream)

InputStream / OutputStream for byte streams; Reader / Writer for character streams.

ii. Reading and Writing Data Using Streams

Use FileInputStream/FileOutputStream for bytes, FileReader/FileWriter or BufferedReader/BufferedWriter for characters, and Scanner for parsing text input.

iii. Handling File I/O Operations

Check for exceptions, use try-with-resources to ensure streams are closed, and consider java.nio.file (Files, Paths) for modern file operations.