MODULE-6 CORE JAVA

1) Introduction to Java

- **History of Java**

  Java was created in the early 1990s by a team of engineers at Sun Microsystems, led by James Gosling. Initially, it was meant for programming household devices and was called **Oak**. But since "Oak" was already trademarked by another company, the name was changed to **Java**, inspired by coffee.

  The language was officially launched in **1995** with the promise of "write once, run anywhere," thanks to the Java Virtual Machine (JVM), which allowed code to run on different platforms without needing changes.

  Over time, Java became popular for web development (especially applets back then), enterprise applications, and later Android apps. In **2010**, Oracle acquired Sun Microsystems and took over Java's development. Since then, the language has evolved with regular updates, adding features like lambda expressions, streams, and modules.

  Today, Java remains one of the most widely used programming languages in the world, known for its stability, cross-platform nature, and large community.

- **Key Features of Java**

  Platform Independent: Java programs run on any system with a Java Virtual Machine (JVM). You write the code once, and it works on Windows, Mac, or Linux without changes.

  Object-Oriented: Everything in Java revolves around classes and objects, which makes it easy to organize and reuse code.

  Simple and Familiar: Java's syntax is clean and somewhat similar to C/C++, but without the confusing parts like pointers.

  Secure: Java runs inside a controlled environment (the JVM), which adds a layer of security. It also has built-in features to avoid common threats like memory leaks.

  Robust: It handles errors through exception handling and has strong memory management (like automatic garbage collection).

Multithreaded: Java makes it easy to run multiple tasks at the same time, using built-in support for threads.

High Performance: Though not as fast as C++, Java is quite efficient thanks to Just-In-Time (JIT) compilation.

Distributed: Java supports networking and remote method invocation (RMI), which makes it great for building distributed systems.

Portable: Java code runs the same way across different machines, as long as they have the JVM installed

- **Understanding JVM, JRE, and JDK**

  JVM (Java Virtual Machine)

  JVM is the engine that runs Java programs. It takes the compiled .class files (bytecode) and executes them. It's what makes Java platform-independent because the same bytecode can run on any machine with a JVM.

  JRE (Java Runtime Environment)

  JRE includes the JVM and the libraries needed to run Java applications. It's what you need if you're only running Java programs, not writing or compiling them.

  JDK (Java Development Kit)

  JDK is for developers. It includes the JRE, plus tools like the compiler (javac), debugger, and other utilities needed to write, compile, and debug Java programs

- **Setting up the Java environment and IDE (e.g., Eclipse, IntelliJ)**

  Setting up an IDE:
  An IDE makes coding easier by providing features like auto-completion, syntax highlighting, debugging tools, and project management. Two popular IDEs for Java are Eclipse and IntelliJ IDEA.

  Eclipse:

  - Download from the official Eclipse website.
  - Install and launch it.

Create a new Java project, write your code in .java files, and run it directly from the IDE.

o Java Program Structure (Packages, Classes, Methods)

- Packages

Packages in Java are like folders on your computer – they help group related classes together.

They prevent name conflicts and make large projects more organized.

There are two types:

Built-in packages (e.g., java.util, java.io)

User-defined packages (created by the programmer)

- **Classes**

  A **class** is the blueprint from which objects are created.

  It contains variables (data) and methods (behavior).

  Every Java program must have at least one class.

- **Methods**

  A **method** is a block of code that performs a specific task.

  Methods help avoid code repetition and improve reusability.

  The main() method is the entry point of every Java program.

2)Data Types, Variables, and Operators

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

In Java, primitive data types are the basic types of data that store simple values directly in memory. There are 8 of them, each with a fixed size and specific purpose.

1. **byte** – 1 byte, range: -128 to 127, used for saving memory in large arrays.
2. **short** – 2 bytes, range: -32,768 to 32,767, rarely used except in memory-constrained systems.
3. **int** – 4 bytes, commonly used for integer values, range: about ±2 billion.
4. **long** – 8 bytes, used for very large integer values, range: very high positive and negative limits.
5. **float** – 4 bytes, used for decimal values with single precision (~7 digits).
6. **double** – 8 bytes, used for decimal values with double precision (~15 digits), default for floating-point numbers.
7. **char** – 2 bytes, stores a single Unicode character.
8. **boolean** – 1 bit (size depends on JVM), stores true or false.

o Variable Declaration and Initialization

Variable Declaration and Initialization

In Java, a variable is a name given to a memory location used to store data. Before using a variable, it must be declared with a specific data type. Declaration tells the compiler what type of data the variable will hold, while initialization assigns an initial value to it.

Example:

- Declaration: int age;
- Initialization: age = 22;
- Both together: int marks = 85;

Types of Variables:

1. **Local Variables** – Declared inside methods, accessible only within them.
2. **Instance Variables** – Declared in a class but outside methods, unique for each object.

3. **Static Variables** – Declared with static keyword, shared by all objects of the class.

Proper declaration and initialization ensure correct data handling and prevent runtime errors like "variable might not have been initialized."

o Operators: Arithmetic, Relational, Logical, Assignment, Unary, and Bitwise

In Java, operators are special symbols used to perform operations on variables and values. They help in performing calculations, comparisons, and logical decisions.

1. Arithmetic Operators
Used for basic mathematical operations.

- + (addition), - (subtraction), * (multiplication), / (division), % (modulus).

2. Relational Operators
Used to compare two values and return a boolean result.

- == (equal to), != (not equal to), > (greater than), < (less than), >= (greater or equal), <= (less or equal).

3. Logical Operators
Used to combine boolean expressions.

- && (logical AND), || (logical OR), ! (logical NOT).

4. Assignment Operators
Used to assign values to variables.

- = (assign), += (add and assign), -= (subtract and assign), *= (multiply and assign), /= (divide and assign), %= (modulus and assign).

5. Unary Operators
Operate on a single operand.

- ++ (increment), -- (decrement), + (positive sign), - (negative sign), ! (logical NOT).

6. Bitwise Operators
Work at the binary (bit) level.

- & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise NOT), << (left shift), >> (right shift), >>> (unsigned right shift)

o Type Conversion and Type Casting

In Java, type conversion and type casting are used to change the data type of a variable or value so that it can be used in a different context.

1. Type Conversion (Widening / Implicit Casting)

- Performed automatically by Java when converting a smaller data type to a larger one.
- No data loss occurs.
- Example: int → long → float → double.
- Also called **widening conversion**.

2. Type Casting (Narrowing / Explicit Casting)

- Done manually by the programmer when converting a larger data type to a smaller one.
- May cause data loss or precision loss.
- Requires casting syntax: (dataType) value.
- Example: double → float → long → int.

Key Points:

- Widening is safe and automatic.
- Narrowing must be done carefully to avoid loss of data.
- Type casting is often used when working with mixed data types in calculations or when handling user input.

Example Flow:
byte → short → int → long → float → double (widening)

3) Control Flow Statements

o If-Else Statements

If-Else Statements
In Java, the **if-else** statement is used for decision-making. It executes a block of code if a condition is true; otherwise, it executes another block.

- **if** – checks a condition.
- **else** – executes when the condition is false.

- **else if** – checks multiple conditions in sequence.

o Switch Case Statements

The **switch** statement is used when we have multiple possible values for a variable and want to execute different code for each case.

- Each case represents a possible value.
- break is used to exit the switch after a case matches.
- default runs if no case matches.
- Works well with byte, short, int, char, String, and enums.

o Loops (For, While, Do-While)

Loops are used to execute a block of code repeatedly.

1. **for loop** – Best when the number of iterations is known.
2. **while loop** – Runs as long as the condition is true, checks the condition before execution.
3. **do-while loop** – Similar to while but checks the condition after running the code at least once.

o Break and Continue Keywords

Break and Continue Keywords

- **break** – Immediately exits from a loop or a switch statement.
- **continue** – Skips the current iteration and moves to the next iteration of the loop.

5) Methods in Java

o Defining Methods

In Java, a method is a block of code that performs a specific task. It is defined using a method name, return type, and parameters (if any). Methods help in reusing code and keeping programs organized.

o Method Parameters and Return Types

Parameters are variables passed into a method to provide input.

Return type specifies the type of value the method will give back. If no value is returned, the type is void.
Example flow: returnType methodName(parameters) { ... return value; }

o Method Overloading

Method overloading means having multiple methods in the same class with the same name but different parameter lists (number or type of parameters).

- Helps improve code readability.
- Decided at compile time (compile-time polymorphism).

o Static Methods and Variables

Static methods belong to the class, not to any object, and can be called using the class name.

Static variables are shared among all objects of a class and have a single memory location.

6. Object-Oriented Programming (OOPs) Concepts

o Basics of OOP: Encapsulation, Inheritance, Polymorphism, Abstraction

Encapsulation – Wrapping data (variables) and code (methods) together inside a class and controlling access using access modifiers.

Inheritance – Acquiring properties and methods of one class (parent) into another class (child) to promote reusability.

Polymorphism – The ability of a method or object to behave in different ways (method overloading and overriding).

Abstraction – Hiding implementation details and showing only the essential features using abstract classes or interfaces.

o Inheritance: Single, Multilevel, Hierarchical

Single Inheritance – One child class inherits from one parent class.

Multilevel Inheritance – A class inherits from another class, which in turn inherits from another (grandparent → parent → child).

Hierarchical Inheritance – Multiple child classes inherit from the same parent class

o Method Overriding

Method Overriding – When a child class defines a method with the same name, return type, and parameters as in the parent class, replacing its implementation.

7) Constructors and Destructors

o Constructor Types (Default, Parameterized)

Default Constructor – A constructor with no parameters. If not defined, Java automatically provides one to initialize objects with default values.

Parameterized Constructor – Accepts parameters to initialize an object with specific values at the time of creation.

o Copy Constructor (Emulated in Java)

Java does not have a built-in copy constructor like C++, but it can be emulated by creating a constructor that takes another object of the same class as a parameter and copies its values.

o Constructor Overloading

Defining multiple constructors in the same class with different parameter lists. This allows creating objects in different ways depending on the data available.

8) Arrays and Strings

- **One-Dimensional and Multidimensional Arrays**

  One-Dimensional Array – A linear collection of elements of the same type, accessed by a single index. Example: int[] arr = new int[5];

  Multidimensional Array – An array containing other arrays, commonly used as 2D arrays (rows and columns). Example: int[][] matrix = new int[3][3];

- **String Handling in Java: String Class, StringBuffer, StringBuilder**

  String Class – Immutable sequence of characters. Once created, values cannot be changed.

  StringBuffer – Mutable (can be modified), thread-safe, suitable for multi-threaded environments.

StringBuilder – Mutable, but not thread-safe; faster than StringBuffer in single-threaded programs.

- **Array of Objects**

  An array can store not only primitive values but also objects. This is useful for storing multiple instances of a class in a single structure. Example: Student[] students = new Student[5];

- **String Methods (length, charAt, substring, etc.)**

  length() – Returns the number of characters in a string.

  charAt(index) – Returns the character at a given index.

  substring(start, end) – Extracts a portion of the string.

  toUpperCase() / toLowerCase() – Changes case of the string.

  equals() – Compares two strings for equality.

9) Inheritance and Polymorphism

Inheritance Types and Benefits

- **Types:**
  1. **Single Inheritance** – One child class inherits from one parent class.
  2. **Multilevel Inheritance** – Inheritance in multiple steps (grandparent → parent → child).
  3. **Hierarchical Inheritance** – Multiple child classes inherit from the same parent class.
- **Benefits:** Code reusability, method and property sharing, easier maintenance, and logical class hierarchy.

Method Overriding

When a subclass provides a new implementation for a method already defined in its superclass with the same name, parameters, and return type. It is used to achieve run-time polymorphism.

Dynamic Binding (Run-Time Polymorphism)

The method call is resolved at runtime based on the object being referenced, not the reference type. This allows different object types to respond differently to the same method call.

Super Keyword and Method Hiding

- **super keyword** – Used to call the parent class constructor, access parent class variables, or invoke parent class methods when overridden.
- **Method Hiding** – When a static method in a subclass has the same signature as a static method in the superclass. It does not override but hides the parent method, and method resolution happens at compile time.

10) Interfaces and Abstract Classes

Abstract Classes and Methods

- An **abstract class** is a class declared with the abstract keyword that cannot be instantiated directly.
- It can contain both abstract methods (without a body) and non-abstract methods (with implementation).
- **Abstract methods** must be implemented by the subclass unless the subclass is also abstract.
- Used when we want to provide a base structure but allow subclasses to define specific behaviors.

Interfaces: Multiple Inheritance in Java

- An **interface** is a collection of abstract methods (and static/default methods) that a class can implement.
- Java does not support multiple inheritance with classes, but a class can implement multiple interfaces, achieving multiple inheritance of type.
- All methods in an interface are public and abstract by default (except static and default methods).

Implementing Multiple Interfaces

- A class can implement more than one interface by separating them with commas.
- The class must provide implementations for all abstract methods of the interface

11) Packages and Access Modifiers

Java Packages: Built-in and User-Defined Packages

- A **package** is a collection of related classes and interfaces, used to organize code and avoid naming conflicts.
- **Built-in packages** are provided by Java, like java.util, java.io, java.sql.
- **User-defined packages** are created by programmers to structure large projects. Example: package mypack;

Access Modifiers: Private, Default, Protected, Public

- **private** – Accessible only within the same class.
- **default** (no modifier) – Accessible within the same package.
- **protected** – Accessible within the same package and by subclasses (even in different packages).
- **public** – Accessible from anywhere in the program.

12) Exception Handling

Types of Exceptions: Checked and Unchecked

- **Checked Exceptions** – Exceptions that are checked at compile time. If not handled, the program will not compile. Examples: IOException, SQLException.
- **Unchecked Exceptions** – Exceptions that occur at runtime and are not checked at compile time. Examples: NullPointerException, ArrayIndexOutOfBoundsException.

try, catch, finally, throw, throws

- **try** – Contains code that might throw an exception.

- **catch** – Handles the exception thrown from the try block.
- **finally** – Contains code that will always run, whether or not an exception occurs (e.g., closing resources).
- **throw** – Used to explicitly throw an exception in code.
- **throws** – Declares the exceptions that a method might throw, so they must be handled by the caller.

Custom Exception Classes

- Java allows creation of user-defined exceptions by extending the Exception class (for checked) or RuntimeException class (for unchecked).
- Useful when we need application-specific error handling.
- Example: class MyException extends Exception { ... }

13) Multithreading

A **thread** is the smallest unit of execution in a program. Java supports multithreading, allowing multiple threads to run concurrently, improving performance and responsiveness (e.g., background tasks while the main task runs).

Creating Threads by Extending Thread Class or Implementing Runnable Interface

1. **Extending Thread class** – Create a class that extends Thread and override the run() method, then call start().
2. **Implementing Runnable interface** – Create a class that implements Runnable, override run(), then pass it to a Thread object and call start().

- Runnable is preferred as it allows extending another class as well.

Thread Life Cycle

1. **New** – Thread object is created but not started.
2. **Runnable** – Thread is ready to run but waiting for CPU scheduling.
3. **Running** – Thread is executing its run() method.
4. **Blocked/Waiting** – Thread is paused waiting for a resource or signal.
5. **Terminated** – Thread has finished execution

15) Collections Framework

The **Collections Framework** in Java is a set of classes and interfaces that provide ready-made data structures and algorithms for storing, manipulating, and retrieving data efficiently. It includes interfaces like List, Set, Map, and implementations like ArrayList, HashSet, etc.

List, Set, Map, and Queue Interfaces

- **List** – Ordered collection that allows duplicate elements. Example: ArrayList, LinkedList.
- **Set** – Unordered collection that does not allow duplicates. Example: HashSet, TreeSet.
- **Map** – Stores data as key-value pairs, keys are unique. Example: HashMap, TreeMap.

ArrayList, LinkedList, HashSet, TreeSet, HashMap

- **ArrayList** – Resizable array, fast for accessing elements.
- **LinkedList** – Doubly linked list, fast for insertions/deletions.
- **HashSet** – Stores unique elements, uses hashing, no order.
- **TreeSet** – Stores unique elements in sorted order.
- **HashMap** – Stores key-value pairs, fast lookups using hashing.

Iterators and ListIterators

- **Iterator** – Used to traverse elements of a collection one by one, supports hasNext(), next(), and remove().
- **ListIterator** – Used to traverse lists in both forward and backward directions, also allows modification of elements.