Module 6 – Core Java

# Q-1.Introduction to Java

## History of Java

Java was developed in the early 1990s by James Gosling and his team at Sun Microsystems under the Green Project. Initially, it was called Oak, but later renamed to Java due to trademark issues. It was officially released in 1995 and became popular for its principle of 'Write Once, Run Anywhere' (WORA), made possible by the Java Virtual Machine (JVM). In 2009, Oracle Corporation acquired Sun Microsystems and continued Java’s development. Today, Java is widely used in web, enterprise, cloud, and mobile (Android) applications.

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

Java has many powerful features that make it popular. It is platform-independent, as bytecode can run on any system with a JVM. Being object-oriented, it supports concepts like inheritance, polymorphism, and encapsulation. It is considered simple and secure, as it avoids complex features like pointers and provides strong runtime security. Java is robust and portable, with automatic garbage collection and error handling. It also supports multithreading, distributed programming, and achieves better performance with the

Just-In-Time (JIT) compiler.

## Understanding JVM, JRE, and JDK

The Java Virtual Machine (JVM) is responsible for running Java bytecode on any platform by converting it into machine code. The Java Runtime Environment (JRE) provides the JVM and libraries required to run Java programs but does not include development tools. The Java Development Kit (JDK) includes the JRE along with development tools like the compiler (javac) and debugger. In short, JDK = JRE + Development tools, and JRE = JVM

+ Libraries. Together, they form the backbone of Java development.

## Setting up the Java Environment and IDE (e.g., Eclipse, IntelliJ)

To set up Java, the JDK must be installed first, and environment variables like PATH and JAVA\_HOME should be configured. This allows execution of Java commands (javac, java) from the command line. For ease of coding, developers use Integrated Development Environments (IDEs) like Eclipse and IntelliJ IDEA, which provide features like code suggestions, debugging, and project management. Eclipse is widely used in enterprises, while IntelliJ is preferred for its smart features and user-friendly interface.

## Java Program Structure (Packages, Classes, Methods)

A typical Java program consists of packages, classes, and methods. A package is a collection of related classes and interfaces, used to organize code. A class is the basic building block containing variables and methods. The program begins execution from the main() method. Methods contain statements that define the behavior of the class. For example, a simple Java program has import statements, a class definition, and the main() method where execution starts.

# Q-2.Data Types, Variables, and Operators

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

Java has eight primitive data types: byte, short, int, long (integer types), float, double (decimal types), char (single character), and boolean (true/false). These are the building blocks of data manipulation and are stored directly in memory for fast access.

## Variable Declaration and Initialization

A variable in Java is a named memory location used to store data. It must be declared with a type before use (e.g., int age;). Variables can be initialized when declared (e.g., int age = 20;). Java supports local variables, instance variables, and static variables depending on their scope and usage.

## Operators in Java : Arithmetic, Relational, Logical, Assignment, Unary, and Bitwise.

Operators are special symbols that perform operations on variables and values. Java provides different types of operators: Arithmetic (+, -, \*, /, %), Relational (==, !=, <, >, <=,

>=), Logical (&&, ||, !), Assignment (=, +=, -=), Unary (++ ,--, +, -), and Bitwise (&, |, ^, ~,

<<, >>).

## Type Conversion and Casting

Type conversion in Java is of two types: implicit (widening) and explicit (narrowing). In implicit conversion, a smaller data type is automatically converted to a larger type (e.g., int to double). In explicit casting, the programmer forces conversion using cast operators (e.g., (int) 10.5). This allows flexibility in handling different types of data.

# Q-3.Control Flow Statements

## If-Else Statements

The if-else statement is a decision-making control structure. It allows execution of a block of code based on a condition. If the condition is true, the if block executes; otherwise, the

else block executes. It helps implement conditional logic in programs.

## Switch Case Statements

The switch statement is used when multiple conditions need to be checked. It evaluates an expression and executes the matching case block. If no case matches, the default block is executed. It is more efficient and readable than multiple if-else statements in certain situations.

## Loops (For, While, Do-While)

Loops allow executing a block of code repeatedly. The for loop is used when the number of iterations is known. The while loop executes as long as the condition is true. The

do-while loop is similar to while but ensures the block executes at least once. Loops are essential for repetitive tasks like traversing arrays.

## Break and Continue Keywords

The break keyword is used to exit a loop or switch immediately. The continue keyword skips the current iteration and proceeds to the next one in a loop. Both are used to control the flow of execution in programs.

# Q-4.Classes and Objects

## Defining a Class and Object in Java

A class is a blueprint that defines the structure and behavior (fields and methods) of objects. An object is an instance of a class created using the new keyword. Classes promote code reusability, and objects represent real-world entities in a program.

## Constructors and Overloading

A constructor is a special method that is automatically invoked when an object is created. It initializes object values. Java supports constructor overloading, where multiple constructors exist with different parameter lists, allowing flexibility in object creation.

## Object Creation and Accessing Members

Objects are created using the new keyword followed by the constructor (e.g., Student s = new Student();). Members of a class, such as variables and methods, are accessed using the dot (.) operator (e.g., s.name, s.display()).

## This Keyword

The this keyword refers to the current object of the class. It is used to differentiate between instance variables and parameters when they have the same name, to call current class methods/constructors, and to pass the current object as an argument.

# Q-5.Methods in Java

**1.Defining Methods:**

In Java, a method is a block of code that performs a specific task and can be reused multiple times. Methods are defined inside classes and help in modular programming by dividing large programs into smaller, manageable parts. A method has a name, a return type, and may accept parameters.

**2.Method Parameters and Return Types**:

Methods can accept input values called parameters, which allow data to be passed into the method. A method may also return a value using the return statement. Methods that don’t return anything use the void keyword.

**3.Method Overloading**:

Method overloading in Java allows multiple methods to have the same name but different parameter lists. It improves code readability and reusability. The compiler differentiates between overloaded methods based on the number or type of parameters.

**4.Static Methods and Variables**:

Static methods and variables belong to the class rather than an object. A static variable is shared by all objects of the class, while static methods can be called without creating an object. They are useful for common operations like utility methods (e.g., Math.sqrt()).

# Q-6.Object-Oriented Programming (OOPs) Concepts

**1.Basics of OOP:**

OOP in Java is a programming paradigm based on objects that combine data and methods. The four main principles are Encapsulation, Inheritance, Polymorphism, and Abstraction.

**2.Inheritance:**

**I**nheritance allows a new class (child) to reuse properties and behaviors of an existing class (parent). Types of inheritance in Java include Single Inheritance, Multilevel Inheritance, and Hierarchical Inheritance.

**3.Method Overriding and Dynamic Method Dispatch:**

Method overriding occurs when a subclass provides a specific implementation of a method already defined in its parent class. Dynamic Method Dispatch is the mechanism through which the method call is resolved at runtime, allowing Java to achieve runtime polymorphism.

# Q-7.Constructors and Destructors

**1.Constructor Types:**

A constructor is a special method in Java used to initialize objects. It has the same name as the class and does not have a return type. Java provides Default Constructors and Parameterized Constructors.

**2.Copy Constructor (Emulated in Java):**

Java does not provide a built-in copy constructor, but it can be emulated by creating a constructor that accepts another object of the same class and copies its values.

**3.Constructor Overloading**:

Constructor overloading allows multiple constructors in the same class with different parameter lists. This provides flexibility in initializing objects in different ways.

**4.Object Life Cycle and Garbage Collection**:

Objects in Java are created using the new keyword and destroyed automatically by the Garbage Collector. Java does not support destructors like C++; instead, it uses automatic memory management.

# Q-8.Arrays and Strings

**1.One-Dimensional and Multidimensional Arrays:**

An array in Java is a collection of elements of the same type stored in contiguous memory locations. A one-dimensional array is like a list, while a multidimensional array (e.g., 2D array) is like a table with rows and columns.

**2.String Handling in Java:**  **String Class, String Buffer, StringBuilder**

Strings in Java are objects of the String class, which is immutable. Java also provides String Buffer and StringBuilder classes for mutable strings.

**3.Array of Objects:**

In Java, arrays can hold not only primitive data but also objects. An array of objects stores references to objects instead of values.

**4.String Methods** **(length, char At, substring, etc.):-**

The String class provides many built-in methods such as length(), charAt(), substring(), equals(), and compareTo(). These methods make string handling efficient and powerful.

**Q-9.Inheritance and Polymorphism**

**1.Inheritance Types and Benefits:**

Inheritance is a mechanism in Java where one class (child or subclass) acquires the properties and behaviors of another class (parent or superclass). The main types of inheritance in Java are single, multilevel, and hierarchical. Java does not support multiple inheritance through classes to avoid ambiguity, but it can be achieved using interfaces. The benefits of inheritance include code reusability, method overriding for polymorphism, and improved code organization, making applications easier to maintain and extend.

**2.Method Overriding:**

Method overriding occurs when a subclass provides its own implementation of a method already defined in the parent class. The method in the child class must have the same name, return type, and parameters as in the parent. Overriding is used to achieve runtime polymorphism and allows subclasses to provide specialized behavior while reusing the parent’s logic. The @Override annotation is often used for clarity.

**3.Dynamic Binding (Run-Time Polymorphism):**

Dynamic binding refers to the process where the method call is resolved at runtime rather than compile-time. In Java, this happens when an overridden method is called through a reference of the parent class but the actual method executed is based on the object’s runtime type. This is the essence of runtime polymorphism and makes programs more flexible and extensible.

**4.Super Keyword and Method Hiding:**

The super keyword in Java is used to access members of the parent class, such as variables, methods, and constructors. It is useful when a child class overrides a method but still needs to call the parent class’s version. Method hiding occurs when a subclass defines a static method with the same signature as a static method in the parent class. Unlike overriding, method hiding is resolved at compile time and depends on the reference type, not the object type.

**Q-10.Interfaces and Abstract Classes**

**1.Abstract Classes and Methods:**

An abstract class in Java is a class declared with the abstract keyword, and it may contain abstract methods (without implementation) along with concrete methods. Abstract classes cannot be instantiated directly; instead, they provide a base for subclasses to extend and implement the abstract methods. They are used when classes share common behavior but also need to provide specific implementations in derived classes.

**2.Interfaces: Multiple Inheritance in Java:**

An interface in Java is a collection of abstract methods (before Java 8) and can also include default, static, and private methods (from Java 8 onwards). Interfaces are used to achieve multiple inheritance since a class can implement multiple interfaces. This allows Java to avoid the “diamond problem” faced in multiple class inheritance. Interfaces define a contract that implementing classes must follow.

**3.Implementing Multiple Interfaces:**

In Java, a single class can implement more than one interface by separating them with commas in the implements keyword. This allows classes to inherit behavior from multiple sources and provides great flexibility in design. However, if multiple interfaces contain default methods with the same signature, the implementing class must override the method to resolve the conflict.

**Q-11: Packages and Access Modifiers**

**1.Java Packages: Built-in and User-Defined Packages**

A package in Java is a way of grouping related classes, interfaces, and sub-packages together, making code easier to organize and maintain. Java provides two types of packages: built-in packages and user-defined packages. Built-in packages are predefined and come with the Java API, such as java.util, java.io, and java.sql, which contain useful classes and methods for various tasks. User-defined packages, on the other hand, are created by programmers to logically group their own classes. This helps avoid naming conflicts, improves reusability, and makes large projects more manageable.

**2.Access Modifiers: Private, Default, Protected, Public**

Access modifiers in Java define the scope or visibility of variables, methods, constructors, and classes. There are four main types: private, default, protected, and public. The private modifier makes a member accessible only within the same class. The default (no modifier) provides access within the same package. The protected modifier allows access within the same package and by subclasses, even if they are in different packages. The public modifier gives the widest access, making the member accessible from anywhere. Using access modifiers correctly ensures data encapsulation and security in Java applications.

**3.Importing Packages and Class Path**

To use classes from a package in Java, the import keyword is used. For example, import java.util.Scanner; imports the Scanner class from the java.util package. We can import a single class, multiple classes, or an entire package using a wildcard (\*). Additionally, Java uses a class path to locate classes and packages at runtime. The class path is an environment variable or command-line option that specifies the directories or JAR files where Java should search for classes. By properly managing imports and the class path, programmers can effectively use built-in and user-defined packages in their projects.

**Q-12. Exception Handling**

In Java, **exception handling** is a mechanism to handle runtime errors, ensuring the normal flow of the program. Exceptions are events that disrupt program execution, such as invalid user input, file not found, or division by zero. They provide a structured way to detect errors and take corrective actions.

**Types of Exceptions:-**

**Checked Exceptions**: These are exceptions that are checked at compile-time by the compiler. If a method can throw a checked exception, it must either handle it using try-catch or declare it with the throws keyword. Examples include IO Exception, SQL Exception, and Class Not Found Exception. They represent conditions that a well-written program should anticipate and recover from.

**Unchecked Exceptions:** These are exceptions that occur at runtime and are not checked at compile-time. They are subclasses of Runtime Exception. Examples include Null Pointer Exception, Arithmetic Exception, and Array Index Out Of Bounds Exception. These usually indicate programming mistakes such as logic errors or improper use of APIs. Thus, exception handling helps in creating robust applications by categorizing exceptions into checked (recoverable) and unchecked (mostly due to coding errors).

**Q-13.Multithreading**

**1. Introduction to Threads**

A thread in Java is the smallest unit of execution within a process. A process can have multiple threads running concurrently, allowing programs to perform multiple tasks at the same time (multithreading). For example, one thread can handle user input while another processes data in the background. Java provides built-in support for multithreading, which improves performance and responsiveness in applications like games, servers, and multimedia programs.

**2. Creating Threads**

There are **two ways** to create threads in Java:

* **Extending the Thread Class**
  + You create a class that extends the Thread class.
  + Override the run() method with the code to execute in the thread.
  + Create an object of your class and call start() to run it.
* **Implementing the Runnable Interface**
  + Create a class that implements the Runnable interface.
  + Implement the run() method with the task code.
  + Pass the object of your class to a Thread object and call start().

The **Runnable approach** is preferred because Java allows multiple inheritance through interfaces, making code more flexible.

**3. Thread Life Cycle**

The life cycle of a thread in Java consists of several states:

1. **New (Created):** Thread object created but not started (Thread t = new Thread();).
2. **Runnable:** After calling start(), the thread is ready but waiting for CPU scheduling.
3. **Running:** The thread is executing its run() method.
4. **Waiting/Timed Waiting:** The thread is temporarily inactive (wait(), sleep(), join()).
5. **Terminated (Dead):** The thread has finished execution and cannot be restarted.

**4. Synchronization and Inter-thread Communication**

* **Synchronization:**  
  When multiple threads share resources (like variables, files, or databases), conflicts may occur. Synchronization ensures that only one thread accesses a shared resource at a time. In Java, this is done using the synchronized keyword on methods or code blocks.
* **Inter-thread Communication:**  
  Java provides mechanisms (wait(), notify(), and notifyAll()) for threads to communicate and coordinate their work. For example, one thread may wait until another thread completes a task, and then it gets notified to continue execution. This avoids busy-waiting and improves efficiency.