Revision of 26th October 2024

1. Introduction to Java

Java is a high-level, object-oriented programming language developed by **Sun Microsystems** (now part of Oracle) in **1995**. It was designed with the concept of **WORA** ("Write Once, Run Anywhere"), meaning that Java applications are platform-independent and can run on any device equipped with a Java Virtual Machine (JVM). Java combines the efficiency of C++ with a simpler, safer design that makes it easier to write and maintain complex applications.

Key Features of Java:

- **Object-Oriented**: Java is based on OOP principles (e.g., encapsulation, inheritance, polymorphism).
- **Platform Independent**: Java code is compiled into bytecode, which can be run on any platform with a JVM.
- **Robust and Secure**: Java has a strong memory management system and includes built-in security features.
- **Multithreaded**: Java has built-in support for multithreaded applications, making it suitable for concurrent programming.

Applications of Java: Java is used to build desktop applications, web applications, mobile applications (primarily Android), enterprise solutions, and IoT (Internet of Things) systems.

---

2. Versions of Java

Java has evolved significantly since its inception, with many versions introducing new features, security enhancements, and performance improvements. The most significant updates came with **Java SE 8, Java SE 11, and Java SE 17**, which are **Long-Term Support (LTS) versions**.

Key Versions and Their Features:

- **Java SE 6**: Introduced scripting language support, JDBC 4.0, and performance improvements.
- **Java SE 7**: Introduced try-with-resources for better resource management, NIO.2, and support for binary literals.
- **Java SE 8**: Added lambda expressions, the Stream API, and default methods in interfaces, improving functional programming capabilities.
- **Java SE 9**: Introduced the Java Platform Module System (Project Jigsaw) and JShell, an interactive REPL.
- **Java SE 10 and beyond**: Each release adds minor updates and features, with Java adopting a **6-month release cycle** to push frequent, incremental improvements.

LTS vs. Non-LTS Versions: LTS (Long-Term Support) versions like **Java 8, 11, and 17** are supported for several years, making them ideal for production. Non-LTS versions receive only short-term support.

---

3. JDK, Compilation and Execution Model, Types of Errors, Install Eclipse

- **Java Development Kit (JDK)**: The JDK is a software development environment that includes the **Java Runtime Environment (JRE)**, Java compiler **(javac),** and other development tools needed to write and run Java programs. The JDK is essential for Java development.
- **Compilation and Execution Model**:
  - **Source Code (.java files)** is written by the programmer.
  - The **Java Compiler** converts this source code into **bytecode (.class files).**
  - The **JVM (Java Virtual Machin**e) executes this **bytecode,** making the application platform-independent.
- **Types of Errors**:
  - **Syntax Errors**: Detected by the compiler; these are issues with code syntax.
  - **Runtime Errors**: Errors that occur during program execution, such as **NullPointerException**.

    - **Logical Errors**: Errors in the logic of the program that produce incorrect output but do not prevent the program from running.
- **Installing Eclipse**: **Eclipse is a popular IDE for Java development**. Installing Eclipse involves downloading the latest version, installing JDK, and configuring the IDE's preferences.

---

4. Eclipse UI Overview

Eclipse User Interface (UI) consists of multiple components:

- **Package Explorer**: Displays the structure of projects, packages, and classes within the workspace.
- **Editor Window**: The main area where you write and edit code.
- **Console**: Displays program output and logs.
- **Outline**: Shows the structure of the active Java file, listing methods and variables for easy navigation.
- **Properties and Problems View**: Displays project properties and compilation errors.

Perspectives in Eclipse:

- **Java Perspective**: Provides views and tools specific to Java development.
- **Debug Perspective**: Contains debugging tools for tracking and fixing issues in the code.

Customizing the Workspace: Users can rearrange and configure views and perspectives to create a personalized development environment.

---

5. Eclipse Shortcuts

Shortcuts in Eclipse:

- **Editing**:
    - **Ctrl + Space:** Code completion.

    - **Ctrl + D:** Delete line.
- **Navigation**:
    - **Ctrl + Shift + T**: Open class or interface.
    - **F3:** Go to declaration.
- **Debugging**:
    - **F5:** Step into a method.
    - **F6:** Step over.
- **Refactoring**:
    - **Alt + Shift + R**: Rename variables, methods, or classes.
    - **Alt + Shift + L**: Extract local variables.

Importance of Shortcuts: Mastering shortcuts can significantly speed up coding and debugging, making developers more efficient.

---

6. Creating a Sample Java Project

Creating a Java project in Eclipse involves:

- **Starting a New Project**: Go to File -> New -> Java Project and specify project settings.
- **Creating a Package and Class**: Add packages to organize code and classes to define the logic.
- **Writing a Basic Java Program**: Writing a "Hello, World!" program to understand syntax and structure.
- **Configuring Build Path**: Includes managing libraries and dependencies to ensure the project compiles and runs correctly.

Project Structure:

- **src**: Holds the source code files.
- **bin**: Holds the compiled **bytecode** files (if auto-build is enabled).

---

7. Running a Project in Eclipse

Running Java Programs:

- **Run Configurations**: Eclipse allows you to configure how a program is run, including JVM arguments, program arguments, and working directories.
- **Console Output**: Eclipse's console displays output, which helps in understanding program flow and debugging.

Debugging Basics:

- **Breakpoints**: Mark specific lines in the code where execution will pause, allowing inspection of variables and program state.
- **Inspecting Variables**: While debugging, view the value of variables to diagnose issues.
- **Stepping Through Code**:
  - **Step Into (F5)**: Enters into a method to see its inner workings.
  - **Step Over (F6)**: Executes the line without going into method details.

---

8. Running a Project Outside Eclipse (Classpath and Comment Lines)

Exporting and Running Java Applications:

- **Exporting a Project as a JAR File**: Java projects can be exported as **JAR (Java Archive) files** for easy sharing and execution outside Eclipse.
- **Running from the Command Line:** Use the command java -**cp <classpath> <MainClass>** to execute a Java application from the terminal. The **classpath** is a parameter that tells the JVM where to find classes and libraries.

Classpath in Java:

- **Classpath**: Specifies directories or JAR files containing classes and packages the program requires. Setting classpath is essential for complex projects with multiple libraries.

Commenting in Code:

- **Single-line Comments** (//): Used for brief comments on individual lines of code.
- **Multi-line Comments** (/* ... */): Used for commenting out multiple lines or providing detailed explanations.
- **Javadoc Comments** (/** ... */): Special comments for generating documentation, often placed above methods and classes to explain their purpose and usage.