Day 1 - Agenda

1. Java Introduction
2. History and Evolution of Java
3. Key Features of Java
4. Understanding the JDK, JRE, and JVM
5. HelloWorld Java Program
6. Data Types
7. Access Modifies
8. Java Naming conventions
9. Class and Object with example

What is Java?

Java is a high-level, object-oriented programming language developed by Sun Microsystems, which is now owned by Oracle Corporation.

When we say Java is a high-level programming language, it means:

1. **Abstraction from Hardware:** Java abstracts away the details of the computer's hardware, making it easier to write and understand code.
2. **Platform Independence:** Java programs are compiled into bytecode that can run on any device with a Java Virtual Machine (JVM), adhering to the "Write Once, Run Anywhere" principle.
3. **Advanced Features:** Java provides extensive libraries and frameworks for various tasks, supports object-oriented programming, and handles memory management automatically.
4. **Ease of Use:** Java has a human-readable syntax, strong error-checking, and powerful development tools that enhance productivity and simplify complex tasks.

When we say Java is an object-oriented programming (OOP) language, it means:

1. **Classes and Objects:** Java uses classes as blueprints to create objects, which are instances containing data and methods.
2. **Encapsulation:** Data and methods are bundled into a single unit (class), restricting direct access to some components.
3. **Inheritance:** New classes can inherit attributes and methods from existing classes, promoting code reuse.
4. **Polymorphism:** Methods can perform different tasks based on the object, allowing one interface to handle multiple implementations.
5. **Abstraction:** Complex details are hidden, exposing only necessary parts, simplifying interaction with objects.

History and Evolution of Java

Origins

- **Creation by Sun Microsystems:** Java was created by Sun Microsystems, an American technology company, in the mid-1990s.
- **James Gosling:** James Gosling, often referred to as the "father of Java," led the development team. The project was initially called "Oak," named after an oak tree that stood outside Gosling's office.
- **Renaming to Java:** The name was later changed to Java, inspired by Java coffee, to avoid trademark issues with Oak Technologies.

Key Milestones

- **1991 - The Green Project:** Java started as part of the Green Project, an initiative aimed at developing software for intelligent consumer electronic devices.
- **1995 - Public Introduction:** Java was officially launched to the public in 1995 with the release of Java 1.0. Sun Microsystems described it as a "Write Once, Run Anywhere" (WORA) language, emphasizing its cross-platform capabilities.
- **1996 - Java Development Kit (JDK) 1.0:** The first official release of JDK 1.0 made Java widely accessible, providing the tools needed for developers to write Java programs.

Evolution Over the Years

- **1997 - Java 1.1:** This version introduced important features like inner classes, JavaBeans, and JDBC (Java Database Connectivity).
- **1998 - Java 2 (J2SE 1.2):** Marking a significant update, Java 2 introduced the Swing graphical API, Collections Framework, and performance improvements.
- **2004 - Java 5 (J2SE 5.0):** This version brought major language enhancements, including generics, metadata (annotations), enumerated types, and the enhanced for loop.
- **2006 - Java 6:** Focused on performance improvements, Web Services enhancements, and scripting language support through JSR 223.
- **2011 - Java 7:** Introduced the try-with-resources statement, the diamond operator, and improved support for dynamic languages.
- **2014 - Java 8:** A major release, Java 8 introduced lambda expressions, the Stream API, and the new Date and Time API, significantly modernizing the language.
- **2017 - Java 9:** Featured the Java Platform Module System (Project Jigsaw), which modularized the JDK and improved scalability and performance.
- **2018 - Java 10 and Beyond:** Java adopted a more rapid release cadence, with new versions every six months, introducing incremental enhancements and new features regularly.
- 2024 - JavaThe latest version of Java is Java 22 or JDK 22 released on March, 19th 2024.

Ownership and Governance

- **2009 - Oracle Acquisition:** Oracle Corporation acquired Sun Microsystems, and with it, the ownership of Java. Oracle continues to maintain and develop the Java language.
- **Java Community Process (JCP):** The JCP oversees the evolution of Java through a community-driven process, allowing stakeholders to contribute to the development of the language and its specifications.

Key Features of Java

1. **Simple:**
    - Java is designed to be easy to learn and use. It removes many complex and error-prone features found in other languages, such as explicit pointers and operator overloading.
2. **Object-Oriented:**
    - Java follows the object-oriented programming (OOP) paradigm, which organizes software design around objects and classes. This promotes modularity, reusability, and ease of maintenance.
3. **Platform-Independent:**
    - Java is "Write Once, Run Anywhere" (WORA). Java code is compiled into bytecode, which can run on any device equipped with a Java Virtual Machine (JVM), making it platform-independent.
4. **Secure:**
    - Java has several built-in security features, including bytecode verification, a security manager, and an extensive set of APIs for encryption and secure communication. These ensure that Java applications are protected from unauthorized access and other security threats.
5. **Robust:**
    - Java emphasizes early error checking, runtime checking, and the elimination of error-prone constructs like pointers. It has strong memory management through automatic garbage collection and exception handling mechanisms.
6. **Multithreaded:**
    - Java has built-in support for multithreading, allowing developers to write programs that can perform many tasks simultaneously. This is useful for applications that require high performance, such as games and web servers.
7. **Architecture-Neutral:**
    - Java's bytecode is designed to be architecture-neutral, meaning that it can be executed on any processor with a suitable JVM, without modification.
8. **Interpreted and Compiled:**
    - Java source code is compiled into bytecode, which is then interpreted or compiled at runtime by the JVM. This hybrid approach provides high performance and portability.

9. **High Performance:**
    - Java's performance is enhanced through Just-In-Time (JIT) compilation, which compiles bytecode into native machine code at runtime, improving execution speed.
10. **Distributed:**
    - Java is designed for the distributed environment of the internet. It includes extensive libraries for networking, remote method invocation (RMI), and supports enterprise-level distributed computing.
11. **Dynamic:**
    - Java is designed to adapt to an evolving environment. It can dynamically load classes, support dynamic linking, and provides runtime type information.
12. **Portable:**
    - Java's portability stems from its platform independence and the ability to run on various hardware architectures. The Java standard libraries ensure consistent behavior across different platforms.
13. **Rich Standard Library:**
    - Java provides a comprehensive standard library that includes APIs for data structures, networking, input/output, graphical user interfaces (GUIs), and more, simplifying development.

Understanding the JDK, JRE, and JVM

1. JDK (Java Development Kit)

The JDK is a complete software development kit required to develop Java applications. It includes the JRE (Java Runtime Environment), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed for Java development.

- **Components of the JDK:**
    1. **Java Compiler (javac):** Converts source code written in Java into bytecode (.class files) that can be executed by the JVM.
    2. **Java Runtime Environment (JRE):** Provides the libraries, Java Virtual Machine (JVM), and other components to run applications written in Java.
    3. **Java Debugger (jdb):** A tool for debugging Java programs.
    4. **Java Archive (jar):** A tool for packaging multiple files into a single archive file.
    5. **Other Tools:** Includes `javadoc` for generating documentation, `javap` for class file disassembler, and more.

2. JVM (Java Virtual Machine)

The JVM is an abstract computing machine that enables a computer to run a Java program. It is the runtime environment for Java bytecode, ensuring that the same code can be executed on any platform that has a compatible JVM.

- **Components of the JVM:**
    1. **Class Loader:** Loads class files into memory. It verifies the correctness of the bytecode and loads classes on demand.
    2. **Bytecode Verifier:** Checks the code fragments for illegal code that can violate access rights to objects.
    3. **Interpreter:** Reads and executes bytecode instructions directly.
    4. **Just-In-Time (JIT) Compiler:** Improves performance by converting bytecode into native machine code at runtime.
    5. **Garbage Collector:** Manages memory by automatically deleting objects that are no longer in use.
    6. **Runtime:** Provides core libraries and other support for running Java applications.

3. JRE (Java Runtime Environment)

The JRE provides the libraries, Java Virtual Machine (JVM), and other components to run applications written in Java. It does not contain development tools like compilers or debuggers. It is the implementation of the JVM.

- **Components of the JRE:**
    1. **Java Virtual Machine (JVM):** The engine that runs the Java bytecode.
    2. **Core Libraries:** Libraries that provide the basic functionality of the Java programming language (e.g., java.lang, java.util).
    3. **Supporting Files:** Configuration files, property files, and other files required to run Java programs.

Detailed Workflow

1. **Writing Code:**
    - Developers write Java source code in `.java` files using a text editor or IDE (Integrated Development Environment).
2. **Compiling Code:**
    - The Java compiler (`javac`) compiles the source code into bytecode, stored in `.class` files. This bytecode is platform-independent.
3. **Class Loading:**
    - The class loader loads the `.class` files into memory. This process includes loading, linking (verification, preparation, and resolution), and initialization.
4. **Bytecode Verification:**
    - The bytecode verifier checks the code to ensure it adheres to Java language specifications and does not violate access restrictions.
5. **Execution:**
    - The JVM interprets the bytecode or compiles it to native machine code using the JIT compiler for execution. The runtime environment provides the necessary resources for the program to run.
6. **Memory Management:**
    - The garbage collector manages memory by automatically deallocating memory occupied by objects that are no longer referenced.