**MODULE 2 OOPJ**

**ASSIGNMENT 1 READING ASSIGNMENT**

1. **Reading Assignment: A Short History of Java**

**Answers:**

* **Java's Purpose**: Initially intended for consumer devices but later became a general-purpose programming language.
* **Platform Independence**: A key feature that made Java revolutionary and highly adaptable across different systems.
* **Timeline**: Remember the key years (1991 - project start, 1995 - official release).
* **Evolution**: Java's evolution from "Oak" to "Java" and how it became integral to the Internet's growth.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. **Reading Assignment: Java Language Features**

**Anwers :**

* **Object-Oriented**: -
  + everything as an object
  + helps in organizing and managing code effectively.
* **Platform-Independent**:
  + write once, run anywhere
  + Java programs are compiled into a form called bytecode, which can run on any device that has a Java Virtual Machine (JVM).
* **Robust and Secure**:
  + Java has strong memory management,
  + features like automatic garbage collection and exception handling.
  + These contribute to Java's reliability and security.
* **Multithreaded**:
  + Java can handle multiple tasks Concurrently within a single program.
* **High Performance**:
  + Just-In-Time (JIT) compiler in Java enhances performance by converting bytecode into native machine code at runtime, making the execution of programs faster.
* **Distributed**:
  + Java is built to support distributed computing, making it suitable for developing applications that run across multiple networked computers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. **Reading Assignment: Which Version of JDK Should I Use?**

**Answers:**

* **LTS Version**:
  + The latest Long-Term Support (LTS) version is generally recommended for most users.
  + It offers extended support, including security updates, making it a stable and reliable choice for production environments.
* **Non-LTS Version**:
  + If you need the latest features and are willing to trade some stability for innovation, you might choose the latest non-LTS version.
* **Considerations**: When choosing a JDK version, consider factors like:
* **Project Stability**: If your project needs long-term stability, an LTS version is preferable.
* **Libraries and Tools**: Ensure that the libraries and tools you plan to use are compatible with the JDK version you choose.
* **Production Environment**: The JDK version should align with the requirements and constraints of your production environment.

1. **Reading Assignment: JDK Installation Directory Structure**

**Anwers:**

* **bin/**: This folder contains executable files, such as javac (Java compiler), java (Java runtime launcher), and javap (Java class file disassembler), which are essential tools for Java development.
* **lib/**: This directory holds libraries required by the JDK, including class files that are used during Java program execution.
* **jre/**: This folder contains the Java Runtime Environment (JRE), which includes the Java Virtual Machine (JVM) and core libraries necessary for running Java applications.
* **include/**: This directory includes header files that are used for interfacing Java with native code, enabling Java programs to interact with code written in languages like C or C++.
* **src.zip**: This file contains the source code of the Java API, which can be useful for developers who want to understand the inner workings of Java's standard libraries.

1. **Reading Assignment: About Java Technology**

**Answers :**

* **Java Development Kit (JDK)**: The JDK is a complete software development kit used for developing Java applications. It includes tools like the compiler, debugger, and other utilities necessary for writing and testing Java code.
* **Java Runtime Environment (JRE)**: The JRE provides the libraries and runtime environment needed to run Java applications. It includes the Java Virtual Machine (JVM) and core libraries, but does not include development tools like the JDK.
* **Java Virtual Machine (JVM)**: The JVM is a key component that makes Java platform-independent. It converts Java bytecode (produced by the JDK) into machine-specific code at runtime, allowing Java applications to run on any device with a JVM.
* **Write Once, Run Anywhere**: Java’s architecture, centered around the JVM, allows developers to write code once and have it run on any platform, making Java a highly versatile and portable technology.

1. **Coding Assignments**

**ANSWER :**

1. **Hello World Program:** Write a Java program that prints "Hello World!!" to the console.

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello World!!");

}

}

**1. public**

* **Meaning**: public is an access modifier. It means that the class or method is accessible from anywhere in the program or by any other classes. In this case, it makes the HelloWorld class and the main method accessible from outside the class.

**2. class**

* **Meaning**: class is a keyword used to define a class in Java. A class is a blueprint for creating objects, and it contains fields (variables) and methods (functions) that define the behaviour of the objects.

**3. HelloWorld**

* **Meaning**: HelloWorld is the name of the class. Class names in Java typically start with a capital letter, and by convention, they should be descriptive of the class's purpose. Here, HelloWorld is the class that contains the code to print "Hello World!!" to the console.

**4. { } (Curly Braces)**

* **Meaning**: The curly braces {} define the scope of the class or method. Everything within the { } belongs to the class HelloWorld or the main method.

**5. public static void main(String[] args)**

* **public**: This access modifier makes the main method accessible from anywhere. It’s required for the JVM to access this method when the program starts.
* **static**: The static keyword means that the method belongs to the class itself rather than an instance of the class. This allows the JVM to invoke the main method without creating an instance of the HelloWorld class.
* **void**: The void keyword indicates that the method does not return any value. The main method is the entry point of the program, and it doesn't need to return anything.
* **main**: This is the name of the method that serves as the entry point for any Java application. When you run a Java program, the JVM looks for the main method to start execution.
* **String[] args**: This is the parameter of the main method. It’s an array of String objects that allows the program to accept command-line arguments. If no arguments are provided, the array is empty.

**6. System.out.println("Hello World!!");**

* **System**: System is a predefined class in the java.lang package that provides access to system-level resources and input/output streams.
* **out**: out is a static member of the System class and is an instance of PrintStream. It represents the standard output stream, typically the console.
* **println**: println is a method of the PrintStream class that prints a line of text to the console. It automatically adds a newline after the output.
* **"Hello World!!"**: This is a string literal that will be printed to the console. The text inside the quotation marks is displayed as-is.

**7. ; (Semicolon)**

* **Meaning**: In Java, the semicolon ; is used to terminate statements. It indicates the end of a complete command.

1. **Compile with Verbose Option:** Compile your Java file using the -verbose option with javac. Check the output.
   * **Command:** javac -verbose HelloWorld.java
   * **Output:** The -verbose option provides detailed information about the classes loaded, the files read, and the resources used during the compilation process.
2. **Inspect Bytecode:** Use the javap tool to examine the bytecode of the compiled .class file. Observe the output.
   * **Command:** javap -c HelloWorld.class
   * **Output:** The javap tool decompiles the bytecode, showing the assembly-like code that the JVM will execute.
3. **Reading Assignment: The JVM Architecture Explained**

* **Class Loader**: The Class Loader is responsible for loading .class files (compiled Java files) into the JVM. It organizes the classes that are needed to run the program.
  + **Example**: Imagine you're opening a book to read different chapters. The Class Loader is like the person who finds and opens each chapter for you.
* **Memory Areas**: The JVM has several memory areas to manage different parts of a program:
* **Method Area**: Stores class structures like method code.
* **Heap**: The Heap is where objects (instances of classes) are stored.
* **Stack**: Each thread in a program has its own Stack, where it keeps track of method calls and local variables.
* **Program Counter (PC) Register**: This keeps track of which instruction in the program is currently being executed.
* **Native Method Stack**: Used for methods written in languages other than Java, like C or C++.
  + **Example**: Think of the Heap as a storage room where you keep items (objects), and the Stack as a workbench where you handle tools (method calls and variables) while working.
* **Execution Engine**: This is the part of the JVM that actually runs the program. It can do this in two ways:
* **Interpreter**: Executes the code line by line, which can be slower.
* **Just-In-Time (JIT) Compiler**: Compiles the bytecode into machine code on the fly, which makes the execution faster.
  + **Example**: The Interpreter is like reading instructions one by one and following them. The JIT Compiler is like translating the entire instruction set into your native language before starting the task, making it quicker to follow.
* **Garbage Collector**: This part of the JVM automatically finds and removes objects in the Heap that are no longer needed, freeing up memory.
  + **Example**: Imagine cleaning up a workspace by throwing away tools and materials you no longer need, so you have more room to work.

1. **Reading Assignment: The Java Language Environment: Contents**

* **Core Libraries**: Java comes with a set of core libraries that provide essential functionalities:
* **Collections**: These libraries help you manage groups of objects, like lists, sets, and maps.
* **I/O (Input/Output)**: These libraries are used for reading from and writing to files, handling data streams, and managing user input/output.
* **Networking**: Provides tools for working with network connections, such as making HTTP requests or handling sockets.
* **Concurrency**: These libraries help you run multiple tasks at the same time, making your application more efficient.
  + **Example**: If you were building a library system, the Collections library would help you manage the list of books, the I/O library would help you save or retrieve book data from a file, and the Networking library could let you connect to an online catalog.
* **Development Tools**: Java provides various tools to help you write, test, and optimize your code:
* **Compilers**: Converts your Java code into bytecode that the JVM can run.
* **Debuggers**: Helps you find and fix errors in your code.
* **Profiling Tools**: Analyzes your program's performance to identify slow or inefficient parts.
  + - **Example**: While developing a Java application, you would write your code and use the compiler to convert it into a format that can run on any machine. If something goes wrong, you can use the debugger to find the exact spot where the error occurs.
* **APIs (Application Programming Interfaces)**: Java offers rich APIs for various types of development:
* **GUI Development**: Tools for creating graphical user interfaces (e.g., buttons, windows).
* **Web Services**: APIs for building and interacting with web services.
* **Database Connectivity**: Tools like JDBC (Java Database Connectivity) allow you to connect and interact with databases.
  + - **Example**: If you’re building a Java application with a user interface, you’d use the GUI API to design it. If the application needs to store data, you’d use the Database Connectivity API to connect to a database.