Assignment 1

Name : Harsh Kumar Bharti

Question and Solutions:

1. **Read the history of java.**

- Sun Microsystem was a Hardware company
- Consumer Devices small devices for them to create a language
- Green Team : James gosling , Patrick Naughton , Mike Sheridan
- Initial Development : 1991
- The " *7 " Device (1992): To showcase the potential of the language.
- Failure of " *7 ": Time-warner denied set-top box OS and video-on-demand technology for demo.
- Breakthrough with the web(1994) : WebRunner (a Web browser) Applet
- Acquisition of Java : Oracle Corporation acquired Sun Microsystems in 2010.
- First Public Version : Java 1.0 : 1996
- SLOGAN : " Write Once , Run Anywhere " WORA

2. **Conceptual diagram of java**.

○Programming Language – It is written by Developers in ‘.java ‘ file ○ Java Compiler – It converts the ‘.java ‘ file into bytecode ‘.class ‘ file .

○ Bytecode (.class) file – It is a platform independent code which can on any machine witha JVM

○ Java Virtual Machine (JVM) – Executes bytecode on different platforms.

○ Java Runtime Environment (JRE) – Provide Standard libraries and APIs necessary for javaApplications to run

3**. Java Version History**.

Java 1.0 (1996): Initial release; introduced JVM and basic libraries.
Java 1.1 (1997): Added inner classes, JDBC, and Java Beans.
Java 1.2 (1998): "Java 2"; introduced Swing, Collections Framework.
Java 1.3 (2000): Improved performance with HotSpot JVM.
Java 1.4 (2002): Added assertions, NIO package.

Java 5 (2004): "Java 1.5"; introduced generics, metadata, enums. Java 6 (2006): Focused on performance, added scripting support.

Java 7 (2011): Added try-with-resources, fork/join framework.

Java 8 (2014): Major update; introduced lambdas, Stream API, new Date/Time API.

Java 9 (2017): Introduced the module system (Project Jigsaw).

Java 10 (2018): Added local variable type inference (var).

Java 11 (2018): LTS release; added HTTP Client API.

Java 12 (2019): Added Shenandoah garbage collector.

Java 13 (2019): Introduced text blocks (preview).

Java 14 (2020): Added records (preview), new null pointer exception messages.

Java 15 (2020): Introduced sealed classes (preview).

Java 16 (2021): Added records (standard) and other language features.

Java 17 (2021): LTS release; introduced sealed classes (standard), pattern matching for instanceof. Java 18 (2022): Added new APIs and enhancements.

Java 19 (2022): Introduced record patterns and virtual threads (preview).

Java 20 (2023): Added pattern matching for switch (standardized), virtual threads enhancements.

Java 21 (2023): LTS release; added scoped values (preview), further enhancements

Software Development Kit (SDK).

The Software Development Kit (SDK) is a collection of tools and libraries that developers use to create applications for specific platforms or frameworks.

Key Components of SDK :

1. Libraries
2. APIs
3. Documentation
4. Development
5. Sample Code
6. Integrated Development Kit

5. Java Development Kit (JDK).

The Java Development Kit (JDK) is a comprehensive package provided by Oracle and other vendors for Java developers.

It includes everything needed to develop Java applications.

1. Java Compiler
2. Java Runtime Environment
3. Java Virtual Machine
4. Standard Libraries
5. Development Tools(javap ,Javadoc ,jar , jdb)

6. Java Runtime Environment (JRE).

The Java Runtime Environment (JRE) is a crucial component of the Java platform, specifically designed to run Java applications.

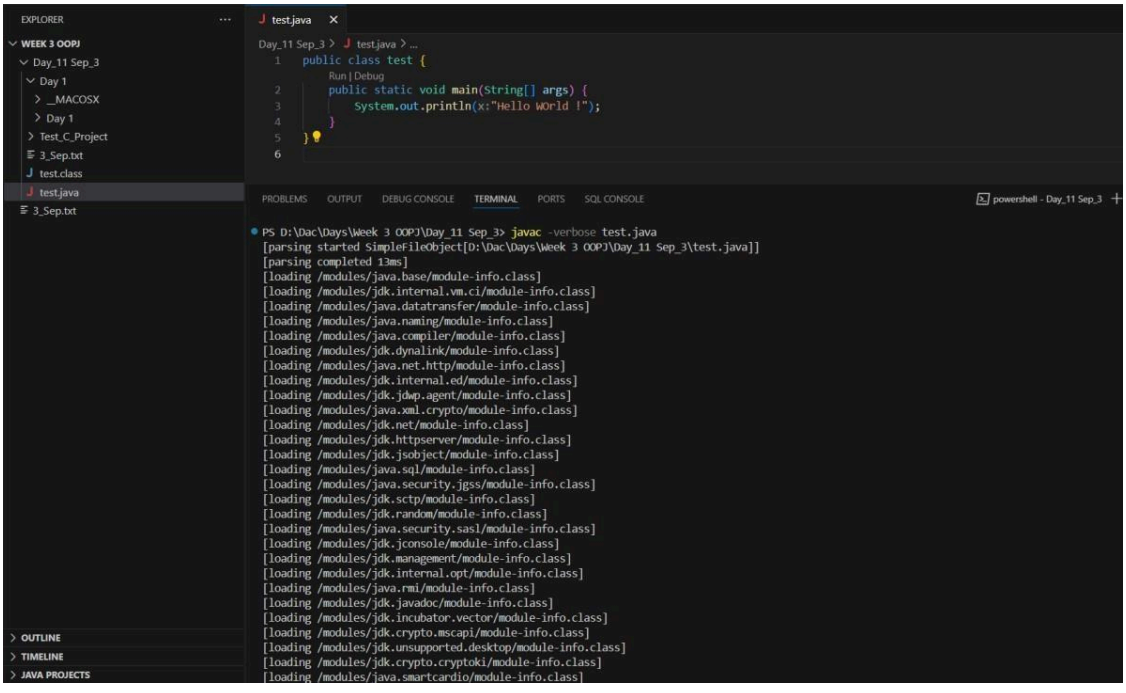
1. JVM
2. Core Libraries
3. Java class Loader
4. Java Runtime Components

7. Hello World program.

```
Class Hello {
        Public static void main (String[]args){
         System.out.println("Hello World!");
        }
}
```

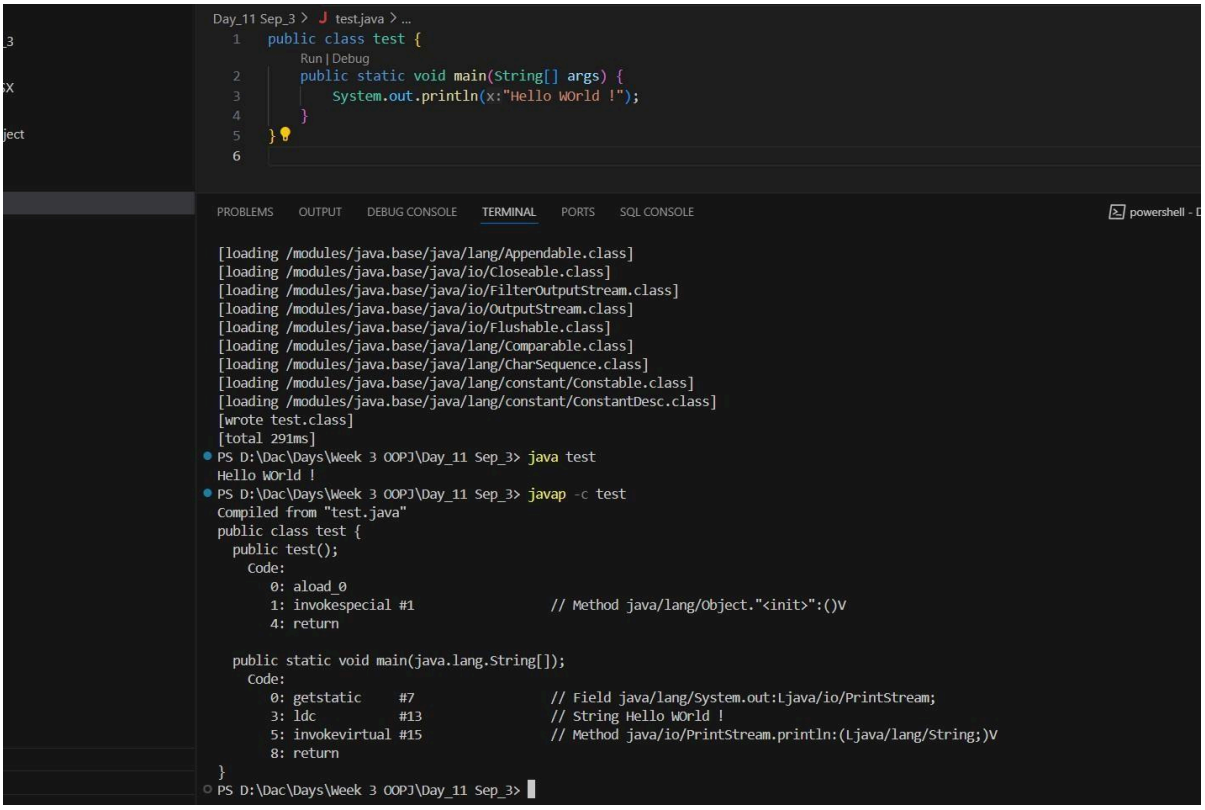
8. Compile with Verbose Option:

Compile your Java file using the *-verbose* option with *javac*.
Check the output.



9. Inspect Bytecode:

Use the javap tool to examine the bytecode of the compiled .class file.
Observe the output.



10. Overview of JVM Architecture.

The Java Virtual Machine (JVM) is a crucial component of the Java programming language. Its primary role is to execute Java bytecode, providing a platform-independent environment for Java applications.

1. Class Loader Subsystem

- **Class Loaders**: These are responsible for loading Java classes into the JVM. The JVM uses a hierarchical class loading system, which includes:
  - **Bootstrap Class Loader**: Loads core Java classes from the JDK's `rt.jar` file.
  - **Platform Class Loader**: Loads classes from the JDK's extension directories.
  - **Application Class Loader**: Loads classes from the application's classpath.
- **Class Loading Process**: This involves locating, loading, and initializing classes. It uses a delegation model where a class loader first delegates the class loading request to its parent loader before attempting to load the class itself.

2. Runtime Data Areas

The JVM divides memory into several runtime data areas used during the execution of a Java program:

- **Method Area**: Stores class-level data, such as runtime constant pool, field and method data, and method and constructor code. This is shared among all threads.

- **Heap**: Used for storing objects and arrays. It is also shared among all threads and is where garbage collection takes place.
- **Stack**: Each thread has its own stack, which contains frames. A frame holds local variables, operand stacks, and dynamic links. The stack is used for method execution and storing method-specific data.
- **Program Counter (PC) Register**: Holds the address of the currently executing instruction. Each thread has its own PC register.
- **Native Method Stack**: Used for native method calls. This stack is specific to the implementation of native methods (methods written in languages other than Java, like C or C++).

3. Execution Engine

The execution engine is responsible for executing the bytecode. It includes:

- **Interpreter**: Reads and executes bytecode instructions one at a time. It's a simple and straightforward way of running bytecode but can be slower compared to other execution methods.
- **Just-In-Time (JIT) Compiler**: Compiles bytecode into native machine code at runtime, which can be executed directly by the host CPU. This improves performance by reducing the need for repeated interpretation of the same code.
- **Garbage Collector**: Automatically manages memory by reclaiming memory occupied by objects that are no longer in use. This helps in preventing memory leaks and optimizing memory usage.

4. Native Interface

- **Java Native Interface (JNI)**: Provides a way for Java code running in the JVM to call and be called by native applications and libraries written in other languages like C or C++. This is useful for leveraging existing libraries or system-level operations not available in Java.

5. Native Method Libraries

- **Java Native Interface (JNI)**: Allows Java to interoperate with native code. The native method libraries are platform-dependent libraries that provide additional functionality to the JVM, like interfacing with hardware or other system resources.

6. Execution Model

- **Thread Management**: The JVM manages multiple threads of execution, each with its own stack. The JVM provides synchronization mechanisms to handle concurrency issues.
- **Security**: The JVM has a built-in security model that includes bytecode verification and a security manager to control access to system resources, ensuring that Java applications run securely.