

Q.1 A Short History of Java

- **Task:** Read about the history and development of Java.
- The history of Java starts with the Green Team. Java team members (Green Team) initiated this project to develop a language for digital devices such as set-top boxes, televisions.
- James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991.
- Java was invented by James Gosling in 1995 at Sun MicroSystem.
- James Gosling firstly it is called as GreenTalk. After that, it was called **Oak** and was developed as a part of the Green project.
- Oak is a symbol of strength and national tree of many countries like U.S.A. , France.
- In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.
- Java is an island in Indonesia where the first coffee was produced (called Java coffee).
- In 1995, Time magazine called Java one of the Ten Best Products of 1995.
- First public Implementation :- JDK 1.0 was released on January 23, 1996.
- Acquisition of Java: Oracle Corporation acquired Sun Microsystems in 2010.
- Slogan: "Write Once, Run Anywhere".

Development Of Java

- Java is built on Object Oriented Principle emphasizing concept like classes, objects, inheritance, polymorphism and encapsulation
- Java's "Write Once, Run Anywhere" philosophy is achieved through the Java Virtual Machine (JVM), which allows Java programs to run on any platform that has a JVM.
- **JDK (Java Development Kit):** The JDK provides tools to develop Java applications, including the Java compiler, Java runtime environment, and various utilities.

Q.2 Java Language Features-

Task: Learn about the main features of Java.

1. **Simple:** Java is designed to be easy to learn and use, with a clear syntax that reduces the complexity of developing applications.
2. **Object-Oriented:** Java follows the object-oriented programming (OOP) paradigm, allowing developers to create modular, reusable code through concepts like classes, objects, inheritance, polymorphism, and encapsulation.
3. **Portable:** Java code is portable across platforms because it is compiled into bytecode, which can be run on any device with a Java Virtual Machine
4. **Platform Independent:** Java is designed to be platform-independent at both the source and binary levels. The motto "Write Once, Run Anywhere" (WORA) signifies that Java programs can run on any device that supports the JVM.
5. **Secured:** Java provides a secure environment for code execution, with built-in features like bytecode verification and runtime security

checks.

6. **Robust:** Java emphasizes reliability and robustness, with strong memory management, exception handling, and type checking mechanisms that minimize the chances of runtime errors and crashes.
7. **Interpreted:** Java is both compiled and interpreted. The Java compiler translates the source code into bytecode, and the JVM interprets this bytecode at runtime.
8. **Distributed:** Java is designed with networking capabilities, making it easy to develop distributed applications. It provides tools for remote method invocation (RMI) and other features to enable communication between computers on a network.

Q.3 Which Version of JDK Should I Use?

Task: Find out which JDK version is right for you.

1. **Long-Term Support (LTS) versions** like JDK 8, 11, or 17 are stable and receive updates for several years.
2. **Latest versions** (e.g., JDK 21) offer new features but have shorter support cycles, suitable for those who need cutting-edge functionalities.

Every two years, the September release will be a Long-Term-Support (LTS) release, which gets updates for at least three years.

| JDK Version | Type | Release Date | Highlights | Recommendation |
|--------------------|---------|--------------|-----------------|--|
| 8 | LTS | 03/2014 | Lambdas | Last LTS version under previous release model. Free updates by Oracle ended , but still maintained by others. Upgrade to 17 or 21 now! |
| 9 | Feature | 09/2017 | Modules | New release model was introduced. EOL. Upgrade to 17 or 21 now! |
| 10 | Feature | 03/2018 | var | EOL. Upgrade to 17 or 21 now! |
| 11 | LTS | 09/2018 | New HTTP Client | Upgrade to 21 now! |
| 12 | Feature | 03/2019 | | EOL. Upgrade to 21 now! |
| 13 | Feature | 09/2019 | | EOL. Upgrade to 21 now! |
| 14 | Feature | 03/2020 | Switch | EOL. Upgrade to 21 now! |

| JDK Version | Type | Release Date | Highlights | Recommendation |
|--------------------|------------|----------------|---|--|
| | | | expressions | |
| 15 | Feature | 09/2020 | Text blocks | EOL. Upgrade to 21 now! |
| 16 | Feature | 03/2021 | Records | EOL. Upgrade to 21 now! |
| 17 | LTS | 09/2021 | Sealed Classes | Supported LTS version. Consider upgrading to 21 in the next months. |
| 18 | Feature | 03/2022 | UTF-8 by Default | EOL. Upgrade to 21 now! |
| 19 | Feature | 09/2022 | | EOL. Upgrade to 21 now! |
| 20 | Feature | 03/2023 | | EOL. Upgrade to 21 now! |
| 21 | LTS | 09/2023 | Pattern Matching , Virtual Threads | Current LTS version. |
| 22 | Feature | 03/2024 | — | Stick with 21. |

Q.4 JDK Installation Directory Structure

- **Task:** Understand the folder structure and files in the JDK installation.

```

jdk-1.8
  bin
    java*
    javac*
    javap*
    javah*
    javadoc*
  lib
    tools.jar
    dt.jar
  jre
    bin
      java*
    lib
      applet
      ext
        jfxrt.jar
        localdata.jar
      fonts
      security
      sparc
        server
        client
      rt.jar
      charsets.jar

```

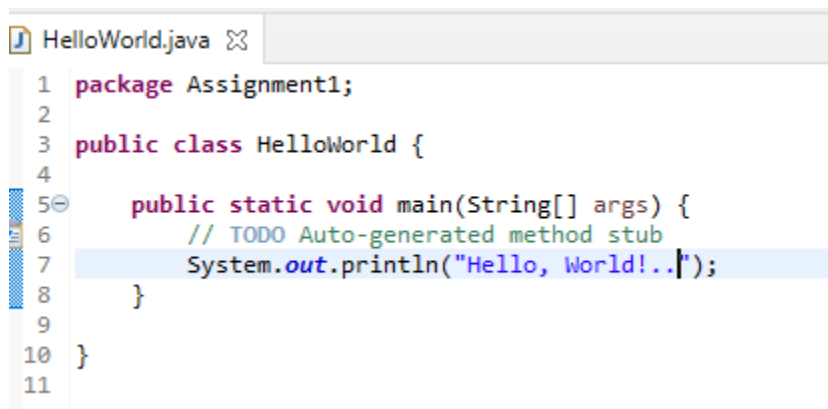
1. **bin/**: Contains executables like javac, java, and javadoc.
2. **lib/**: Holds core libraries and resources needed by the JVM and other tools.
3. **include/**: Contains C/C++ header files for JNI (Java Native Interface).
4. **jre/**: The runtime environment with its own bin/ and lib/ folders.
5. **conf/**: Configuration files for security, networking, etc.

Q. 5 About Java Technology

- **Task:** Read about the basics of Java technology and its components.
- Java technology is both a programming language and a platform.
 - Java technology is a robust, high-level, object-oriented programming language and platform that enables developers to build and deploy applications across various environments.
- The core components of Java technology include:
1. **Java Development Kit (JDK)**: A toolkit that provides the necessary tools, libraries, and utilities for developing Java applications, including the Java compiler (javac) and tools for debugging and documentation.
 2. **Java Runtime Environment (JRE)**: Provides the libraries, Java Virtual Machine (JVM), and other components to run applications written in Java.
 3. **Java Virtual Machine (JVM)**: An abstract computing machine that enables Java bytecode to be executed on any device or operating system, making Java platform-independent. The JVM interprets the compiled bytecode and translates it into machine code that can be executed by the host system.
 4. **Java Standard Edition (Java SE)**: The core Java platform that provides the essential libraries and APIs for general-purpose programming.
 5. **Java Enterprise Edition (Java EE)**: Built on top of Java SE, Java EE provides additional libraries and APIs for building large-scale, distributed, and multi-tiered enterprise applications.
 6. **Java Micro Edition (Java ME)**: A subset of Java SE, designed for resource-constrained devices like embedded systems, mobile phones, and other small devices.

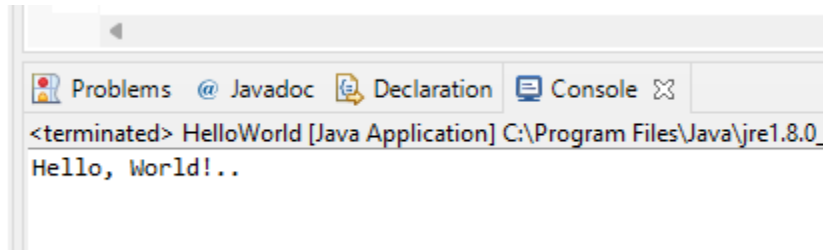
Q. 6 Coding Assignments

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



```
1 package Assignment1;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         System.out.println("Hello, World!!");
8     }
9
10 }
11
```

Output



2. **Compile with Verbose Option:** Compile your Java file using the -verbose option with javac. Check the output.

```
C:\Users\amini\Desktop\CDAC Mumbai\OOPJ Module 2>javac -verbose Hello.java
[parsing started RegularFileObject[Hello.java]]
[parsing completed 16ms]
[search path for source files: .]
[search path for class files: C:\Program Files\Java\jdk1.8.0_202\jre\lib\resources.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\rt.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\sunrsasign.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\jsse.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\jce.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\charsets.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\jfr.jar,C:\Program Files\Java\jdk1.8.0_202\jre\classes,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\access-bridge-64.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\clrdtd.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\dnssns.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\jaccess.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\jfxrt.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\localedata.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\nashorn.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\sunec.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\sunec_provider.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\sunmscapi.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\sunpkcs11.jar,C:\Program Files\Java\jdk1.8.0_202\jre\lib\ext\zipfs.jar,.]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Object.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/String.class)]]
[checking Hello]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/io/Serializable.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/AutoCloseable.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Byte.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Character.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Short.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Long.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Float.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Integer.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Double.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Boolean.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Void.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/System.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/io/PrintStream.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Appendable.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/io/Closeable.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/io/FilterOutputStream.class)]]
[loading ZipFileIndexFileObject[C:\Program Files\Java\jdk1.8.0_202\lib\ct.sym(META-INF/sym/rt.jar/java/io/OutputStream.class)]]
```

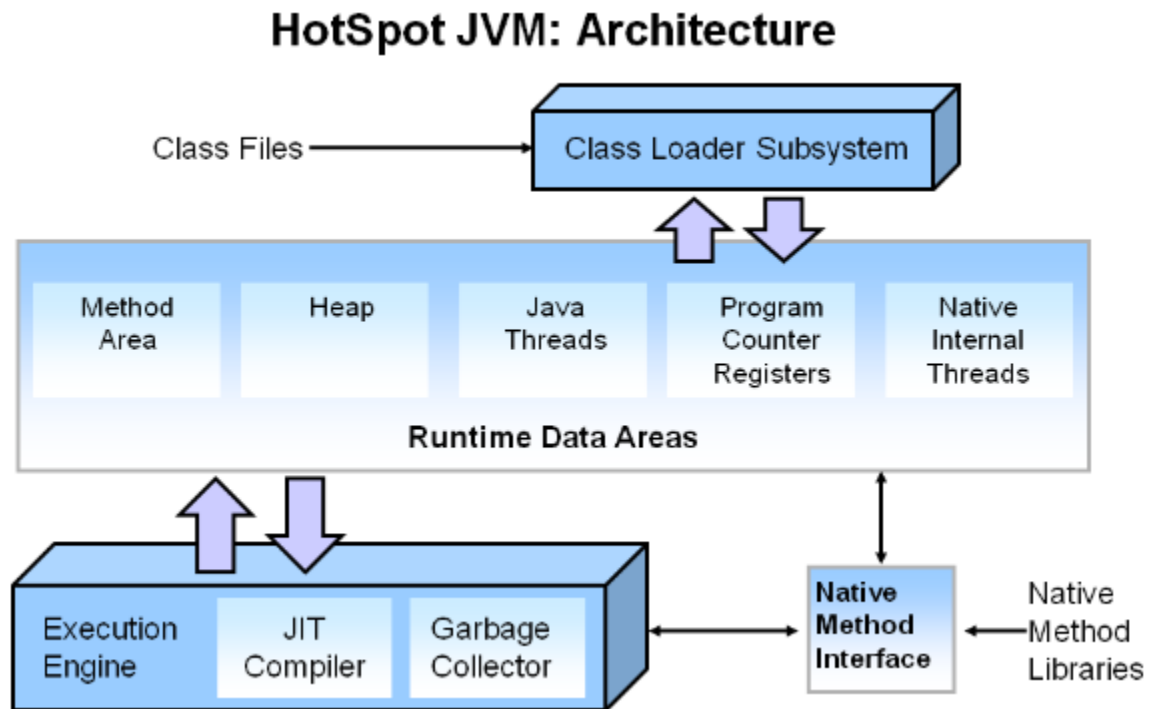
3. **Inspect Bytecode:** Use the javap tool to examine the bytecode of the compiled .class file. Observe the output

```
C:\Users\amini\Desktop\CDAC Mumbai\OOPJ Module 2>javap -c Hello
Compiled from "Hello.java"
class Hello {
    Hello();
    Code:
        0: aload_0
        1: invokespecial #1                  // Method java/lang/Object."<init>":()V
        4: return

    public static void main(java.lang.String[]);
    Code:
        0: getstatic     #2                  // Field java/lang/System.out:Ljava/io/PrintStream;
        3: ldc           #3                  // String Hello World
        5: invokevirtual #4                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        8: return
}
```

Q. 7 The JVM Architecture Explained

- **Task:** Learn about how the Java Virtual Machine (JVM) works.



1. Classloader Subsystem of the JVM

- Classloader is a subsystem of the JVM. Classloader is used to load class files. It verifies class files using a bytecode verifier. A class file will only be loaded if it is valid.

2. Runtime Data Areas of JVM

- Method Area

The method area is also called the class area. The method area stores data for each and every class, like fields, constant pools, and method data and information.

- Heap

The heap is the place where all objects are stored in JVM. The heap even contains arrays because arrays are objects.

- Java Threads (Java Thread Stacks)

Every thread has its own stack. How are stack frames created when threads call new methods? As we know, each and every thread has its own stack. Whenever a new method is called, a new stack frame is created, and it is pushed on top of that thread's stack.

- Program Counter Registers (PC Registers)

The program counter registers contain the address of the instructions currently being executed and the address of next instruction as well.

- Native Internal Threads (Native Thread Stack)

Native internal threads contain all the information related to native platforms. For example, if we're running the JVM on Windows, it will contain Windows-related information. Likewise, if we're running on Linux, it will have all the Linux-related information we need.

3 Execution Engine

The Execution Engine contains the JIT (Just In Time) Compiler and Garbage Collector compiler, as well as the Interpreter

-JIT Compiler

The JIT Compiler compiles bytecode to machine code at runtime and improves the performance of Java applications.

- Garbage Collector

Garbage collection is the process by which the JVM clears objects (unused objects) from the heap to reclaim heap space.

-Interpreter

Interpreter is responsible for reading the bytecode and then executing the instructions.

4. Native Method Libraries of the JVM

The native method interface is an interface that connects the JVM with the native method libraries for executing native methods.

If we are running the JVM (a Java application) on Windows, then the native method interface will connect the JVM with the Windows method libraries for executing Windows methods.

Q.8 The Java Language Environment: Contents

- **Task:** Explore the content and features of the Java language environment.

The Java Language Environment includes a rich set of features such as platform independence through the JVM, strong memory management via automatic garbage collection, extensive libraries, and APIs for various tasks (e.g., networking, I/O, and concurrency). It supports multithreading, security through a sandbox model, and dynamic linking, allowing classes to be loaded and updated at runtime. These features together create a robust and secure environment for developing and running Java applications across different platforms.