

# Assignment No 1

## 1. A Short History of Java

- **key points**

1. **Creation:** Java was developed by James Gosling and the Green Team at Sun Microsystems in the early 1990s and released in 1995.
2. **Initial Name:** Originally called Oak, later renamed Java.
3. **Platform Independence:** Java's "write once, run anywhere" feature is enabled by the Java Virtual Machine (JVM).
4. **Popularity:** Gained traction for web applications and enterprise software due to its cross-platform capabilities.
5. **Oracle Acquisition:** Oracle acquired Sun Microsystems in 2010 and continued to improve Java.
6. **Continued Relevance:** Java remains widely used in industries like finance, cloud computing, and mobile development, known for scalability and reliability.

## 2. Java Language Features

- **Early Versions (Java 1.0 - 1.4):** Java's foundation was laid with core object-oriented features, platform independence, and the Java Virtual Machine (JVM). Early enhancements introduced important features like inner classes, JDBC for database connectivity, RMI for distributed applications, and improvements to the I/O system, including NIO and regular expressions. Java also became more reliable and robust with features like assertions, exception chaining, and logging.

- **Java 5 (2004):** A landmark release, introducing key language features such as generics, annotations, enhanced for loops, and autoboxing/unboxing, which simplified coding and improved type safety.

- **Java 6 (2006) - Java 7 (2011):** Focused on performance improvements and adding new APIs. Key features included scripting support, a more efficient JVM, and functional improvements such as the try-with-resources statement and diamond operator, enhancing the developer experience.

- **Java 8 (2014):** Another pivotal release that introduced functional programming through lambda expressions, the Streams API for handling collections in a functional style, and a new date and time API, greatly enhancing Java's versatility.

- **Java 9 (2017) - Java 11 (2018):** Introduced significant changes like the module system (Project Jigsaw) to improve modularity, JShell (REPL), and new HTTP client APIs. Java 11 became a Long-Term Support (LTS) release, establishing it as a stable version with extended support.

- **Java 12 - Java 15 (2019-2020):** Java focused on preview features such as text blocks, records, pattern matching, and new garbage collectors like Shenandoah and ZGC, which improved performance and memory management.
- **Java 16 - Java 20 (2021-2023):** These versions finalized several preview features, including records and pattern matching, and introduced virtual threads to improve concurrency handling. UTF-8 became the default character set.
- **Java 21 (2023):** Another LTS release, finalizing important features like virtual threads and pattern matching, which further enhanced Java's performance, scalability, and ease of use for modern applications.

### 3. Reading Assignment: Which Version of JDK Should I Use?

#### LTS (Long-Term Support) Versions

- ✓ Recommended: Opt for a long-term support (LTS) version for stability and support. As of now, JDK 8, JDK 11, and JDK 17 are LTS versions.
- ✓ LTS versions get extended support, including security updates, for several years.

#### Latest Features & Stability

- ✓ Latest Features: Non-LTS versions (e.g., JDK 20, 21) come with the newest features but may not have long-term support. These are ideal if you're working on cutting-edge projects or want to explore new features.
- ✓ Stability: LTS versions prioritize stability over new features, making them more suitable for production environments.

### 4. Reading Assignment: JDK Installation Directory Structure

- bin/: Contains executable files for compiling, running and documenting Java programs.
- lib/: Contains required libraries and class files required by the JDK
- include/: Provide header files for writing native code using JNI
- jre/: Stores the Java Runtime Environment, which contains its own executables and libraries for running Java applications.
- src.zip: ZIP file containing source code for standard Java libraries. Useful for learning and debugging.
- conf/: Stores configuration and tools files for the JVM.
- jmods/: Contains Java module files for modular development .
- Legal/: Contains legal and licensing information for JDK components.

### 5. Reading Assignment: About Java Technology

Java is a versatile and widely adopted programming language and platform known for its "Write Once, Run Anywhere" capability. It enables the development of applications that can

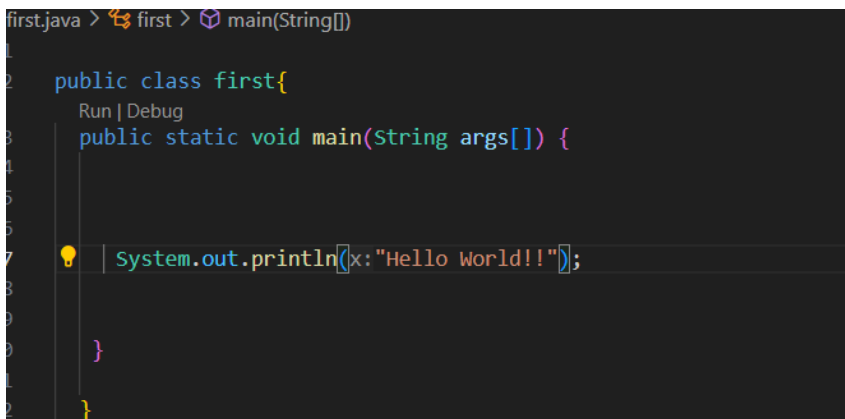
run on any system with a Java Virtual Machine (JVM), ensuring platform independence. Java is object-oriented, secure, and simple to learn, with a focus on creating modular and reusable code. Its memory management system, including automatic garbage collection, ensures efficient use of resources.

## Java Components

- **JDK (Java Development Kit):** A development environment for building Java applications. It includes the JRE, compilers, and development tools.
- **JRE (Java Runtime Environment):** Provides the runtime environment to execute Java applications, including the JVM and core libraries.
- **JVM (Java Virtual Machine):** Interprets and runs Java bytecode on different platforms, making Java platform-independent.

## 6. Coding Assignments

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

A screenshot of a Java IDE with a dark theme. The top bar shows the file path 'first.java' and the class name 'first' with a 'main(String[])' method signature. The code editor displays the following Java code:

```
1 public class first{
2     Run | Debug
3     public static void main(String args[]) {
4
5
6
7     System.out.println("Hello World!!");
8
9     }
10 }
```

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

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

```
PS C:\Users\lenovo\Desktop> java -verbose first.java
[parsing started RegularFileObject[first.java]]
[parsing completed 31ms]
[search path for source files: .]
[search path for class files: C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\resources.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\rt.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\sunrsasign.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\jsse.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\jce.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\narsets.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\jfr.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\rtda.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\ext\access-bridge-64.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\access.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\ext\localedata.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\ext\nashorn.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\ext\suncsc.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\ext\sunec.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\ext\sunpkcs11.jar;C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\jre\lib\ext\zipfs.jar, ]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Object.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/String.class)]]
[checking first]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/io/Serializable.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/AutoCloseable.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Byte.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Character.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Short.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Long.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Float.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Integer.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Double.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Boolean.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Void.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/System.class)]]
[class]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/io/OutputStream.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/io/FileHandle.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/Comparable.class)]]
[loading ZipfileIndexFileObject[C:\Program Files\Eclipse Adoptium\jdk-8.0.422.5-hotspot\lib\ct.sym(META-INF/sym/rt.jar/java/lang/CharSequence.class)]]
[wrote RegularFileObject[first.class]]
[totat 409ms]
```

```

PS C:\Users\Lenovo\Desktop\java> javap -c first
Compiled from "first.java"
public class first {
    public first();
        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

```

## 7. Reading Assignment: The JVM Architecture Explained

Java Virtual Machine (JVM) is an abstract computing machine that enables a computer to run Java programs. The JVM plays a crucial role in running Java programs, as it converts Java bytecode into machine code that can be executed by the host operating system. The JVM is divided into three main subsystems:

### Class Loader Subsystem

The Class Loader subsystem handles dynamic class loading, which means classes are loaded at runtime instead of compile time. It has three main parts:

- Bootstrap : Loads core Java classes from the `rt.jar` file. It has the highest priority.
- Extension : Loads classes located in the `ext` directory.
- Application : Loads classes defined by the user in the application classpath.

Class loading involves three steps:

- Loading: Class files are loaded into the memory.
- Linking: This includes verification (validating bytecode), preparation (allocating memory for static variables), and resolution (replacing symbolic references with actual references).
- Initialization: Static variables are assigned original values, and static blocks are executed.

### Runtime Data Area

This part manages memory during the execution of Java programs and is divided into five areas:

- Method Area: Stores class-level data like static variables. It is shared among all threads.

- **Heap Area:** Stores objects and their instance variables. It is also shared but not thread-safe.
- **Stack Area:** Each thread has its own stack where local variables are stored. It is thread-safe.
- **PC Registers:** Each thread has its own PC register, which stores the address of the currently executing instruction.
- **Native Method Stacks:** Stores native method information for each thread.

### **3. Execution Engine**

The Execution Engine is responsible for executing the bytecode. It has several components:

- **Interpreter:** Executes bytecode line-by-line but is relatively slow because repeated method calls require re-interpretation.
- **Just-In-Time (JIT) Compiler:** Improves performance by compiling repeated bytecode into native code, which is reused for subsequent calls.
- **Garbage Collector:** Automatically reclaims memory by removing unreferenced objects, though calling `System.gc()` does not guarantee immediate execution.

### **8. Reading Assignment: The Java Language Environment: Contents**

- **Java Language Specification:** Defines the syntax and rules for writing and executing Java code.
- **Java SE (Standard Edition):** Core platform for developing and running desktop/server applications, including the JRE and JDK.
- **Java EE (Enterprise Edition :** Now Jakarta EE, adds tools for building large-scale enterprise applications, like web services.
- **Java ME (Micro Edition):** A lightweight version of Java for mobile and embedded devices.
- **Java FX\*\*:** Framework for creating modern, rich user interfaces for internet applications.
- **Java Virtual Machine (JVM):** Executes bytecode, enabling platform independence, memory management, and threading.
- **Java Development Kit (JDK):** Includes tools like the compiler and JRE for developing Java applications.
- **Java Runtime Environment (JRE):** Contains the JVM and libraries to run Java apps..