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

* **Task**: Read about the history and development of Java.
* **Link**: <http://sunsite.uakom.sk/sunworldonline/swol-07-1995/swol-07-java.html>

 **Origins (1991)**: Java's origins trace back to a project called "Green Project" led by James Gosling, Mike Sheridan, and Patrick Naughton at Sun Microsystems. They initially aimed to create software for interactive television, but the project shifted focus towards a platform-independent language.

 **Initial Release (1995)**: Java 1.0 was officially released in 1995. It introduced the concept of "Write Once, Run Anywhere" (WORA), emphasizing portability across different operating systems. Its key features included a simple syntax, object-oriented design, and robust security.

 **Growth and Popularity (Late 1990s)**: Java gained popularity rapidly due to its use in web development with the introduction of Java applets and server-side applications. The Java 2 platform, released in 1998, marked significant advancements, including the introduction of Swing for GUI applications and the Java 2 Enterprise Edition (J2EE) for enterprise solutions.

 **Evolution (2000s)**: Java continued to evolve with new versions introducing improvements and features such as generics, annotations, and the Java Virtual Machine (JVM) enhancements. The release of Java 5 (2004) and Java 6 (2006) were notable milestones.

 **Oracle Acquisition (2010)**: In 2010, Oracle Corporation acquired Sun Microsystems, gaining control over Java. This transition included stewardship of Java's development and ongoing support.

 **Modern Java (2010s-Present)**: Java has continued to evolve with regular updates. The introduction of the Java SE 8 release in 2014 brought significant features like lambdas and the Streams API. Subsequent versions, including Java 9 (2017) and Java 11 (2018), introduced modularity, local-variable type inference (var), and long-term support (LTS) versions. The release cycle has since shifted to a more predictable, six-month release cadence.

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

* **Task**: Learn about the main features of Java.
* **Link**: <https://javaalmanac.io/features/>

 **Object-Oriented**: Java is designed around the concept of objects and classes, promoting modularity, reuse, and flexibility. Key principles include encapsulation, inheritance, and polymorphism.

 **Platform Independence**: Java’s "Write Once, Run Anywhere" (WORA) capability is achieved through the Java Virtual Machine (JVM). Java code is compiled into bytecode, which can run on any device equipped with a JVM, regardless of the underlying hardware and operating system.

 **Simple and Familiar Syntax**: Java's syntax is designed to be easy to learn for programmers familiar with C and C++. It eliminates some of the more complex features of C++, such as multiple inheritance and operator overloading.

 **Strongly Typed**: Java is a strongly typed language, which means that types are checked at compile-time and runtime. This helps catch errors early in the development process and contributes to program stability.

 **Automatic Memory Management**: Java includes an automatic garbage collection mechanism that helps manage memory allocation and deallocation, reducing the likelihood of memory leaks and improving overall program efficiency.

 **Multithreading Support**: Java provides built-in support for multithreading, allowing concurrent execution of multiple threads. This is crucial for developing high-performance applications and handling multiple tasks simultaneously.

 **Robust Standard Library**: Java offers a comprehensive standard library (Java Standard Edition API) that provides a wide range of pre-built classes and methods for tasks like file I/O, networking, data structures, and graphical user interfaces.

 **Security**: Java has a strong security model that includes features like the Java sandbox, bytecode verification, and a security manager. These features help protect against malicious code and ensure safe execution of Java applications.

 **Network-Centric**: Java was designed with networking in mind, and it includes extensive libraries for network communication. Java's networking APIs make it easier to develop distributed and network-based applications.

 **Portability**: Due to its platform-independent nature, Java applications can run on any device that has a compatible JVM. This makes Java highly portable across different computing environments.

 **Dynamic and Extensible**: Java supports dynamic class loading and reflection, allowing applications to be more adaptable and extensible at runtime. This is useful for applications that need to be modified or extended without stopping or recompiling.

 **Rich API and Ecosystem**: In addition to the standard library, Java has a rich ecosystem of frameworks, tools, and libraries that enhance its capabilities, such as Spring for enterprise applications, Hibernate for object-relational mapping, and Apache Maven for project management.

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

* **Task**: Find out which JDK version is right for you.
* **Link**: <https://whichjdk.com/>

For most applications, it's advisable to use a Long-Term Support (LTS) version because these receive extended support and updates, providing stability and security over a longer period. Java 8, while still widely used, is no longer receiving public updates, which may make it less secure compared to newer versions. Java 11, another LTS release, offers a good balance of stability and modern features, including the HTTP Client API and improved performance. However, for new projects or those looking to stay updated with the latest features, Java 17 is a strong choice as it is the most recent LTS version and includes advanced features like sealed classes and pattern matching.

Non-LTS versions, like Java 18 or 19, offer the latest features and enhancements but are supported for a shorter period (typically six months). They are suitable for exploring new technologies or for projects that can accommodate frequent updates. If your application demands stability and long-term support, opting for an LTS version like Java 11 or 17 would be prudent. Conversely, if staying on the cutting edge is essential and you can manage frequent upgrades, experimenting with the latest non-LTS versions might be beneficial.

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

* **Task**: Understand the folder structure and files in the JDK installation.
* **Link**: <https://docs.oracle.com/javase/8/docs/technotes/tools/windows/jdkfiles.html>

 **bin**: This directory contains the executable files for various Java tools and utilities. Key files include javac (the Java compiler), java (the Java runtime), javadoc (the documentation generator), and javap (the class file disassembler). These tools are essential for compiling, running, and managing Java programs.

 **lib**: This directory holds various libraries and configuration files used by the JDK. Notable subdirectories and files include:

* **rt.jar**: This archive contains the runtime classes for the Java platform.
* **tools.jar**: This file contains additional tools and utilities that support development, such as the compiler and the Java Virtual Machine (JVM) tools.

 **include**: This directory contains header files used for native code development. These headers are essential when you are interfacing Java with native applications via Java Native Interface (JNI).

 **jre**: This directory contains a Java Runtime Environment (JRE) which is a subset of the JDK and includes the JVM, core libraries, and other runtime components. It allows running Java applications without the need for the full JDK. This directory mimics the structure of a standalone JRE installation.

 **conf**: This directory includes configuration files for the JDK. It often contains configuration settings for the Java runtime environment and tools.

 **docs**: Some JDK distributions include a docs directory that contains API documentation and other resources for developers.

 **demo**: This directory may contain sample applications and demos showcasing the capabilities of the JDK. These examples are helpful for learning and understanding different features of the Java platform.

 **man**: This directory includes manual pages for various Java tools and commands, similar to man pages in Unix-like systems.

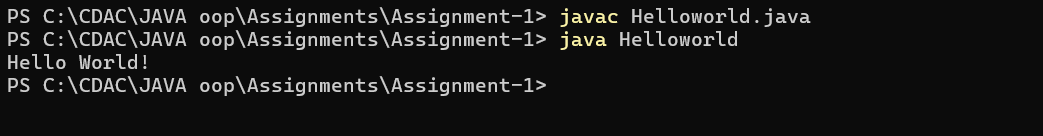
**5. Reading Assignment: About Java Technology**

* **Task**: Read about the basics of Java technology and its components.
* **Link**: <https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>

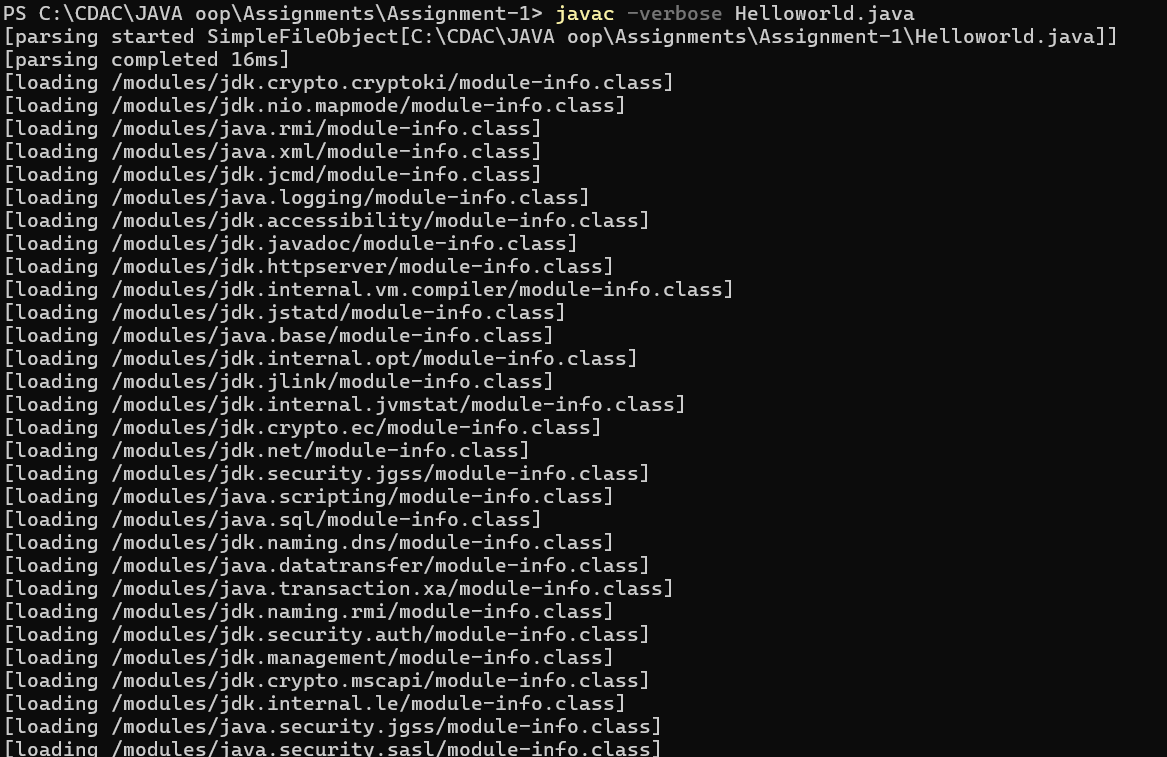
Java technology is a versatile and widely-used programming platform developed by Sun Microsystems, now owned by Oracle. It consists of the Java programming language, the Java Standard Edition (SE), which provides core libraries and tools, and the Java Virtual Machine (JVM), which enables Java applications to run on various platforms without modification. Java's key features include platform independence, object-oriented design, robust security, and automatic memory management. It's used across various domains, from web and mobile applications to large-scale enterprise systems, thanks to its extensive ecosystem and support for multiple frameworks and libraries.

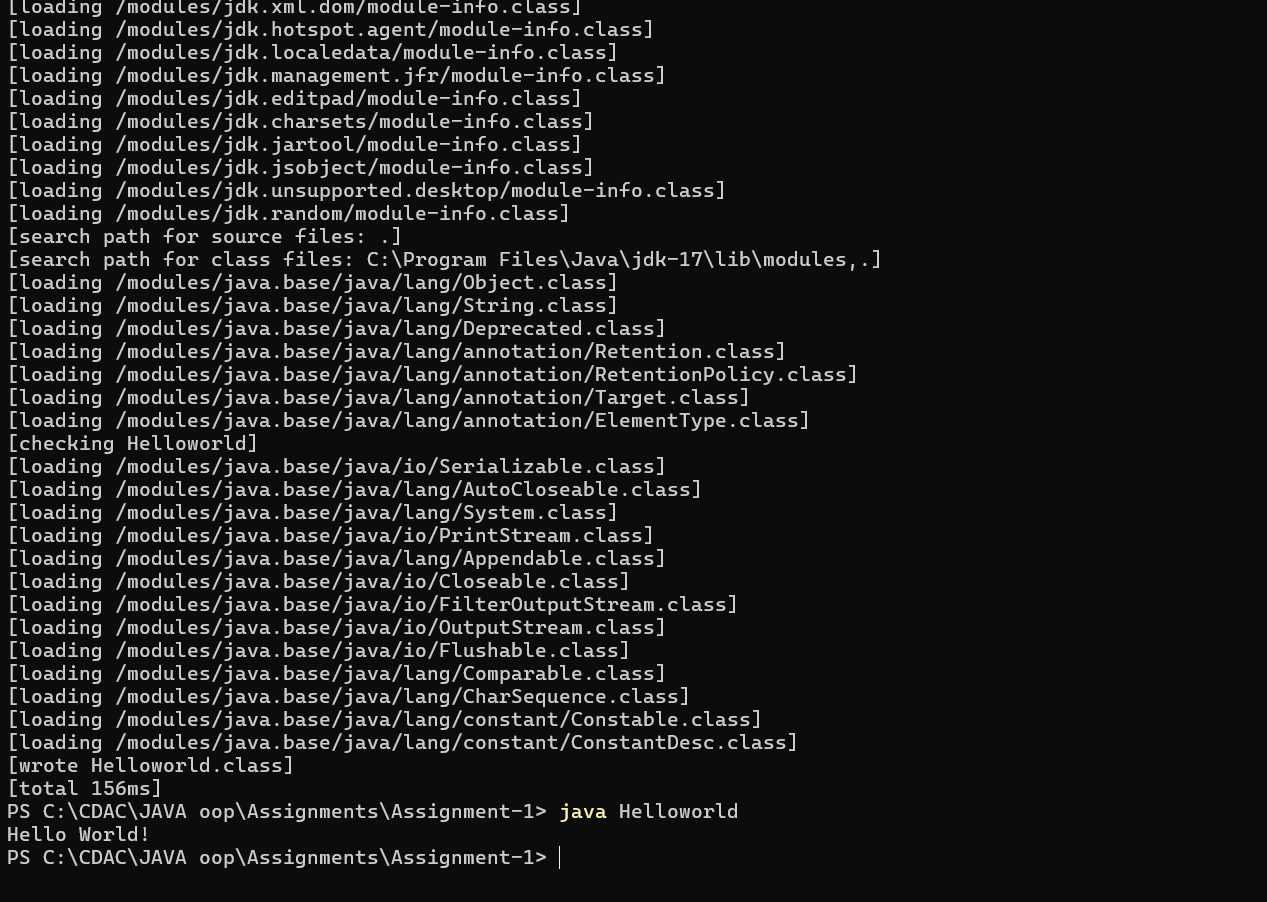
**6. Coding Assignments**

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



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





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

Compiled from "Helloworld.java"

public class Helloworld {

public Helloworld();

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**

* **Task**: Learn about how the Java Virtual Machine (JVM) works.
* **Link**: <https://dzone.com/articles/jvm-architecture-explained>

The Java Virtual Machine (JVM) architecture is designed to execute Java bytecode and ensure platform independence. At its core, the JVM consists of several key components:

The **Class Loader Subsystem** loads, verifies, and initializes Java classes at runtime. It handles different class loaders to manage classes in the hierarchy, including the bootstrap, extension, and application class loaders. The **Runtime Data Areas** include the heap, where objects and class metadata reside; the stack, which holds method call frames; and the method area, which stores class structures and static variables. The **Execution Engine** executes bytecode instructions via an interpreter or Just-In-Time (JIT) compiler, which translates bytecode into native machine code for better performance. The **Garbage Collector** automatically manages memory by reclaiming space from objects that are no longer in use, preventing memory leaks. Finally, the **Native Interface** allows Java programs to interact with native libraries and applications through the Java Native Interface (JNI). This architecture ensures that Java applications run consistently across different platforms by abstracting the underlying hardware and operating system details.

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

* **Task**: Explore the content and features of the Java language environment.
* **Link**: https://www.oracle.com/java/technologies/language-environment.html
* [**Introduction to Java**](https://www.oracle.com/java/technologies/introduction-to-Java.html#318)
  + [Beginnings of the Java Language Project](https://www.oracle.com/java/technologies/introduction-to-Java.html#943)
  + [Design Goals of Java](https://www.oracle.com/java/technologies/introduction-to-Java.html#334)
    - [Simple, Object Oriented, and Familiar](https://www.oracle.com/java/technologies/introduction-to-Java.html#349)
    - [Robust and Secure](https://www.oracle.com/java/technologies/introduction-to-Java.html#367)
    - [Architecture Neutral and Portable](https://www.oracle.com/java/technologies/introduction-to-Java.html#379)
    - [High Performance](https://www.oracle.com/java/technologies/introduction-to-Java.html#1751)
    - [Interpreted, Threaded, and Dynamic](https://www.oracle.com/java/technologies/introduction-to-Java.html#528)
  + [The Java Platform--a New Approach to Distributed Computing](https://www.oracle.com/java/technologies/introduction-to-Java.html#937)
* [**Java--Simple and Familiar**](https://www.oracle.com/java/technologies/simple-familiar.html#343)
  + [Main Features of the Java Language](https://www.oracle.com/java/technologies/simple-familiar.html#1225)
    - [Primitive Data Types](https://www.oracle.com/java/technologies/simple-familiar.html#376)
    - [Arithmetic and Relational Operators](https://www.oracle.com/java/technologies/simple-familiar.html#406)
    - [Arrays](https://www.oracle.com/java/technologies/simple-familiar.html#410)
    - [Strings](https://www.oracle.com/java/technologies/simple-familiar.html#414)
    - [Multi-Level Break](https://www.oracle.com/java/technologies/simple-familiar.html#429)
    - [Memory Management and Garbage Collection](https://www.oracle.com/java/technologies/simple-familiar.html#2333)
    - [The Background Garbage Collector](https://www.oracle.com/java/technologies/simple-familiar.html#455)
    - [Integrated Thread Synchronization](https://www.oracle.com/java/technologies/simple-familiar.html#457)
  + [Features Removed from C and C++](https://www.oracle.com/java/technologies/simple-familiar.html#4076)
    - [No More Typedefs, Defines, or Preprocessor](https://www.oracle.com/java/technologies/simple-familiar.html#4078)
    - [No More Structures or Unions](https://www.oracle.com/java/technologies/simple-familiar.html#4083)
    - [No Enums](https://www.oracle.com/java/technologies/simple-familiar.html#5627)
    - [No More Functions](https://www.oracle.com/java/technologies/simple-familiar.html#5642)
    - [5No More Multiple Inheritance](https://www.oracle.com/java/technologies/simple-familiar.html#4090)
    - [No More Goto Statements](https://www.oracle.com/java/technologies/simple-familiar.html#4093)
    - [No More Operator Overloading](https://www.oracle.com/java/technologies/simple-familiar.html#4098)
    - [No More Automatic Coercions](https://www.oracle.com/java/technologies/simple-familiar.html#4100)
    - [No More Pointers](https://www.oracle.com/java/technologies/simple-familiar.html#4107)
  + [Summary](https://www.oracle.com/java/technologies/simple-familiar.html#4130)
* [**Java is Object Oriented**](https://www.oracle.com/java/technologies/object-oriented.html#343)
  + [Object Technology in Java](https://www.oracle.com/java/technologies/object-oriented.html#2414)
  + [What Are Objects?](https://www.oracle.com/java/technologies/object-oriented.html#1354)
  + [Basics of Objects](https://www.oracle.com/java/technologies/object-oriented.html#6681)
    - [Classes](https://www.oracle.com/java/technologies/object-oriented.html#1372)
    - [Instantiating an Object from its Class](https://www.oracle.com/java/technologies/object-oriented.html#3121)
    - [Constructors](https://www.oracle.com/java/technologies/object-oriented.html#4517)
    - [Methods and Messaging](https://www.oracle.com/java/technologies/object-oriented.html#4559)
    - [Finalizers](https://www.oracle.com/java/technologies/object-oriented.html#2653)
    - [Subclasses](https://www.oracle.com/java/technologies/object-oriented.html#2600)
    - [Java Language Interfaces](https://www.oracle.com/java/technologies/object-oriented.html#6185)
    - [Access Control](https://www.oracle.com/java/technologies/object-oriented.html#6711)
    - [Packages](https://www.oracle.com/java/technologies/object-oriented.html#1389)
    - [Class Variables and Class Methods](https://www.oracle.com/java/technologies/object-oriented.html#2996)
    - [Abstract Methods](https://www.oracle.com/java/technologies/object-oriented.html#3057)
  + [Summary](https://www.oracle.com/java/technologies/object-oriented.html#2314)
* [**Architecture Neutral, Portable, and Robust**](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#319)
  + [Architecture Neutral](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#397)
    - [Byte Codes](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#402)
  + [Portable](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#269)
  + [Robust](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#882)
    - [Strict Compile-Time and Run-Time Checking](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#370)
  + [Summary](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#367)
* [**Interpreted and Dynamic**](https://www.oracle.com/java/technologies/interpreted-dynamic.html#283)
  + [Dynamic Loading and Binding](https://www.oracle.com/java/technologies/interpreted-dynamic.html#1167)
    - [The Fragile Superclass Problem](https://www.oracle.com/java/technologies/interpreted-dynamic.html#1170)
    - [Solving the Fragile Superclass Problem](https://www.oracle.com/java/technologies/interpreted-dynamic.html#1173)
    - [Run-Time Representations](https://www.oracle.com/java/technologies/interpreted-dynamic.html#1183)
  + [Summary](https://www.oracle.com/java/technologies/interpreted-dynamic.html#1918)
* [**Security in Java**](https://www.oracle.com/java/technologies/security-in-java.html#283)
  + [Memory Allocation and Layout](https://www.oracle.com/java/technologies/security-in-java.html#2762)
  + [Security Checks in the Class Loader](https://www.oracle.com/java/technologies/security-in-java.html#2801)
  + [The Byte Code Verification Process](https://www.oracle.com/java/technologies/security-in-java.html#1056)
    - [The Byte Code Verifier](https://www.oracle.com/java/technologies/security-in-java.html#1063)
  + [Security in the Java Networking Package](https://www.oracle.com/java/technologies/security-in-java.html#1461)
  + [Summary](https://www.oracle.com/java/technologies/security-in-java.html#2513)
* [**Multithreading**](https://www.oracle.com/java/technologies/multithreading.html#283)
  + [Threads at the Java Language Level](https://www.oracle.com/java/technologies/multithreading.html#1217)
  + [Integrated Thread Synchronization](https://www.oracle.com/java/technologies/multithreading.html#1222)
  + [Multithreading Support--Conclusion](https://www.oracle.com/java/technologies/multithreading.html#1229)
* [**Performance and Comparisons**](https://www.oracle.com/java/technologies/performance-comparisons.html#2376)
  + [Performance](https://www.oracle.com/java/technologies/performance-comparisons.html#2489)
  + [The Java Language Compared](https://www.oracle.com/java/technologies/performance-comparisons.html#1834)
  + [A Major Benefit of Java: Fast and Fearless Prototyping](https://www.oracle.com/java/technologies/performance-comparisons.html#2451)
  + [Summary](https://www.oracle.com/java/technologies/performance-comparisons.html#1902)
* [**Java Base System and Libraries**](https://www.oracle.com/java/technologies/java-base-system-libraries.html#942)
  + [Java Language Classes](https://www.oracle.com/java/technologies/java-base-system-libraries.html#942)
  + [Input Output Package](https://www.oracle.com/java/technologies/java-base-system-libraries.html#1227)
  + [Utility Package](https://www.oracle.com/java/technologies/java-base-system-libraries.html#1310)
  + [Abstract Window Toolkit](https://www.oracle.com/java/technologies/java-base-system-libraries.html#1392)