**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>
* As Java creator James Gosling explained in a recent interview with SunWorld Online, the genesis of Sun's Web-enhancing technology can be traced to early 1991, when a small group of Sun engineers formed to explore opportunities in the [consumer electronics](http://sunsite.uakom.sk/sunworldonline/swol-07-1995/swol-07-java.html" \l "Lessons) market. At the time, the World Wide Web was still in the drawing rooms.

1. **Origins and Creation (1990-1995)**:

* **Early Beginnings**: Java's development began in the early 1990s by Sun Microsystems. The project was initially known as "Oak" and was spearheaded by James Gosling, Mike Sheridan, and Patrick Naughton. The language was intended for use in interactive television, but it was soon realized that it had broader applications.
* **Rebranding**: In 1995, Oak was renamed "Java." This change was part of a strategic move to better reflect its purpose and market potential.

2. **Java 1.0 and Initial Release (1996)**:

* **Public Release**: Java 1.0 was officially released in 1996. It was designed to be a platform-independent language that could run on any device, leveraging the "Write Once, Run Anywhere" (WORA) philosophy.
* **Java Virtual Machine (JVM)**: A key component of Java’s design was the JVM, which allowed Java applications to run on any platform that had a compatible JVM implementation.

3. **Growth and Evolution (1997-2005)**:

* **Java 2 (J2SE)**: Released in 1998, Java 2 introduced significant enhancements, including the Swing graphical API, Collections Framework, and the Java Plug-in for web browsers.
* **Enterprise and Mobile**: The development of J2EE (Java 2 Enterprise Edition) and J2ME (Java 2 Micro Edition) expanded Java’s reach into enterprise and mobile applications, respectively.

4. **Modern Java (2006-Present)**:

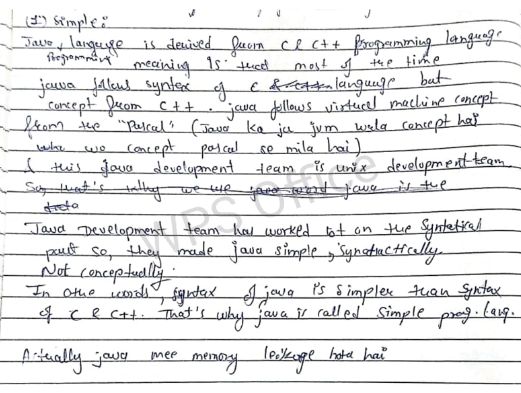
* **Sun Acquired by Oracle**: In 2010, Oracle Corporation acquired Sun Microsystems. Oracle continued to develop and release new versions of Java.
* **Java 5 and Beyond**: The release of Java 5 in 2004 brought major updates, including generics, metadata annotations, and enumerated types. Subsequent versions have introduced new features like lambdas, streams, and the module system.
* **Java Platform, Standard Edition (Java SE)**: Java SE continues to evolve with regular updates, enhancing language features, performance, and security.

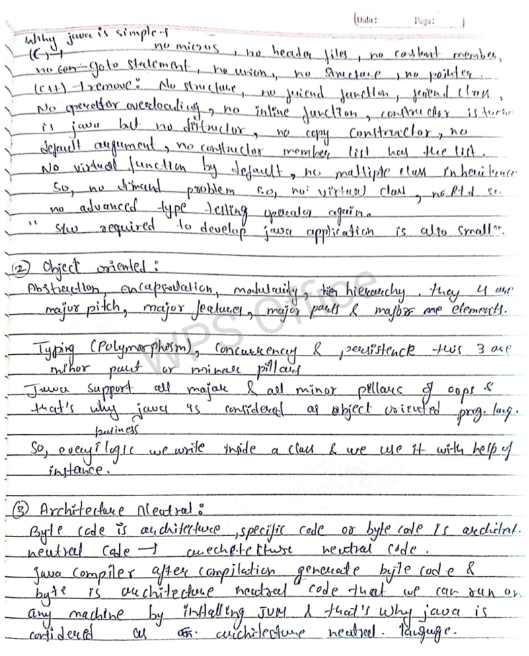
**Key Features of Java**

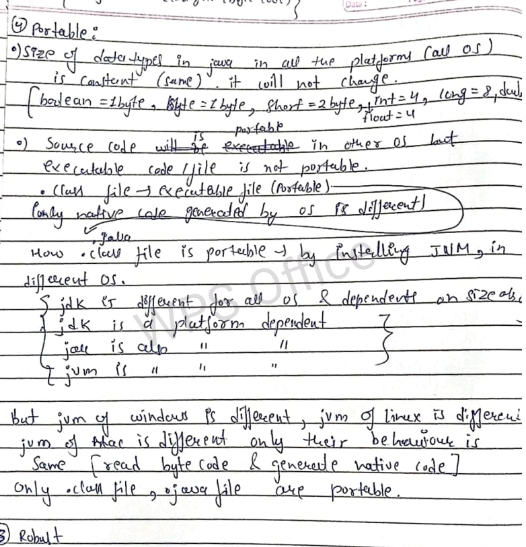
* **Platform Independence**: Java applications are designed to run on any device with a compatible JVM.
* **Object-Oriented**: Java is an object-oriented programming language, which helps in creating modular and reusable code.
* **Automatic Memory Management**: Java provides automatic garbage collection, which helps in managing memory and reducing memory leaks.
* **Rich API**: Java offers a comprehensive set of libraries and APIs for various programming needs, from networking to user interface development.

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

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







 **Platform Independence**

* **Write Once, Run Anywhere (WORA)**: Java is designed to be platform-independent at both the source and binary levels. This is achieved through the use of the Java Virtual Machine (JVM), which allows Java programs to run on any device that has a compatible JVM.

 **Object-Oriented Programming (OOP)**

* **Encapsulation**: Java supports encapsulation, which means that data (attributes) and methods (functions) are bundled together into classes.
* **Inheritance**: Java allows for class hierarchies where one class (subclass) inherits properties and behavior from another class (superclass).
* **Polymorphism**: Java supports polymorphism, enabling objects to be treated as instances of their parent class rather than their actual class.
* **Abstraction**: Java uses abstraction to hide complex implementation details and expose only the necessary parts of the object.

 **Automatic Memory Management**

* **Garbage Collection**: Java includes an automatic garbage collection mechanism that reclaims memory by removing objects that are no longer in use, helping to manage memory more efficiently.

 **Robust Exception Handling**

* **Exception Handling**: Java has a robust exception handling mechanism using try, catch, finally, and throw statements. This helps in managing runtime errors and ensuring the program’s stability.

 **Rich Standard Library**

* **Java API**: Java provides a comprehensive set of standard libraries and APIs for a wide range of functionalities, including data structures, networking, I/O operations, GUI development, and more.

 **Multi-threading**

* **Concurrency Support**: Java supports multi-threading, allowing multiple threads to run concurrently within a program. This is essential for building high-performance and responsive applications.

 **Security**

* **Sandboxing**: Java has built-in security features that include bytecode verification and a security manager that restricts the operations that Java programs can perform.
* **Cryptography**: Java provides extensive libraries for encryption, decryption, and other cryptographic operations.

 **Simplicity and Ease of Use**

* **Clean Syntax**: Java’s syntax is straightforward and resembles C++, making it easier for developers familiar with these languages to learn and use Java.
* **No Pointers**: Java does not support pointers, which eliminates certain types of programming errors and enhances security.

 **Portability**

* **Bytecode Compilation**: Java source code is compiled into bytecode, which is platform-independent. The bytecode is executed by the JVM, making Java applications portable across different operating systems.

 **Network-Centric**

* **Network Programming**: Java has built-in support for network programming through APIs like java.net, making it easier to develop networked applications.

 **Modern Language Features**

* **Lambda Expressions**: Introduced in Java 8, lambda expressions provide a concise way to express instances of single-method interfaces (functional interfaces) and facilitate functional programming.
* **Streams API**: Also introduced in Java 8, the Streams API allows for functional-style operations on collections of elements, such as map-reduce transformations.
* **Modules**: The Java Platform Module System (JPMS), introduced in Java 9, allows developers to create modular applications and manage dependencies more effectively.

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

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

**JDk version**

* **Java 8 (LTS)**: A very stable version widely used in the industry. Includes significant features like lambda expressions and the Streams API.
* **Java 11 (LTS)**: Another LTS version with long-term support. Includes features from Java 9 and 10 and continues to be a popular choice.
* **Java 17 (LTS)**: The most recent LTS version, including features from Java 9 to 16. Recommended for new projects needing long-term support.
* **Java 20** (or newer versions): Includes the latest features and enhancements but may not have long-term support. Ideal for trying out the newest features or for projects that need the latest improvements.

**Conclusion**

Selecting the right JDK version depends on your specific needs, including project requirements, support policies, and compatibility

.

**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>

#### 1. bin **Directory**

* **Purpose**: Contains executable files (binaries) for various Java tools and utilities.
* **Key Files**:
  + javac – The Java compiler.
  + java – The Java runtime launcher.
  + javadoc – The tool for generating API documentation.
  + javap – The class file disassembler.
  + jar – The tool for packaging Java applications into JAR files.
  + keytool – The tool for managing keystores and certificates.

#### 2. lib **Directory**

* **Purpose**: Contains library files and core JDK libraries.
* **Key Files**:
  + rt.jar – The runtime library that contains core Java classes.
  + tools.jar – The library that includes tools and utilities used for development.
  + jce.jar – The Java Cryptography Extension library.
* **Subdirectories**:
  + ext – Contains extension libraries that can be dynamically loaded by the JVM.
  + jfr – Contains files related to Java Flight Recorder.

#### 3. include **Directory**

* **Purpose**: Contains header files used for JNI (Java Native Interface) and other native methods.
* **Key Files**:
  + jni.h – The JNI header file used for native method development.
  + jni\_md.h – Platform-specific JNI header files.

#### 4. jre **Directory**

* **Purpose**: Contains a minimal runtime environment for running Java applications.
* **Key Files**:
  + bin – Contains the java and javaw executables for running Java applications.
  + lib – Contains libraries required to run Java applications.
  + lib/rt.jar – The runtime classes used by the Java runtime.

#### 5. docs **Directory (if included)**

* **Purpose**: Contains documentation related to the JDK.
* **Key Files**:
  + index.html – The main page for the documentation.

#### 6. man **Directory (Unix/Linux)**

* **Purpose**: Contains manual pages for various Java commands and tools.

#### 7. sample **Directory**

* **Purpose**: Contains sample code and examples demonstrating various Java features and tools.

#### 8. src.zip

* **Purpose**: Contains the source code for the Java platform classes. This can be useful for debugging and understanding the implementation of Java standard libraries.

#### 9. **Other Configuration Files**

* **java.security** – Contains security-related configuration settings.
* **java.policy** – Contains policy files for security permissions.

**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>

**Basics of Java Technology**

\*\*1. **What is Java?**

* **Java** is a high-level, class-based, object-oriented programming language designed to be platform-independent. Its core philosophy is “Write Once, Run Anywhere” (WORA), meaning that Java applications can run on any device with a compatible Java Virtual Machine (JVM).

\*\*2. **Key Components of Java Technology**

* **Java Development Kit (JDK)**: The JDK is a software development kit used to develop Java applications. It includes the Java compiler (javac), Java runtime environment, and various tools for development.
* **Java Runtime Environment (JRE)**: The JRE provides the runtime environment necessary to run Java applications. It includes the JVM and the core libraries but does not include development tools like the compiler.
* **Java Virtual Machine (JVM)**: The JVM is the engine that executes Java bytecode. It provides platform independence by abstracting the underlying operating system and hardware.

\*\*3. **Java Language Features**

* **Object-Oriented**: Java is based on object-oriented principles such as inheritance, encapsulation, polymorphism, and abstraction.
* **Platform-Independent**: Java code is compiled into bytecode, which runs on the JVM regardless of the underlying hardware and operating system.
* **Automatic Memory Management**: Java has built-in garbage collection that automatically manages memory allocation and deallocation.

\*\*4. **Java API**

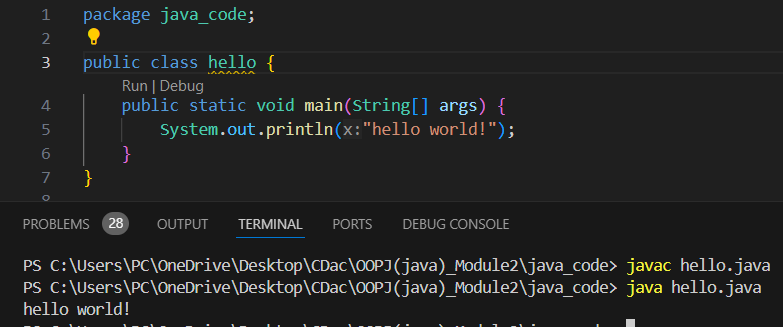
* **Standard Libraries**: Java provides a rich set of libraries and APIs for various functionalities, including networking, I/O operations, and GUI development.

\*\*5. **Development and Execution**

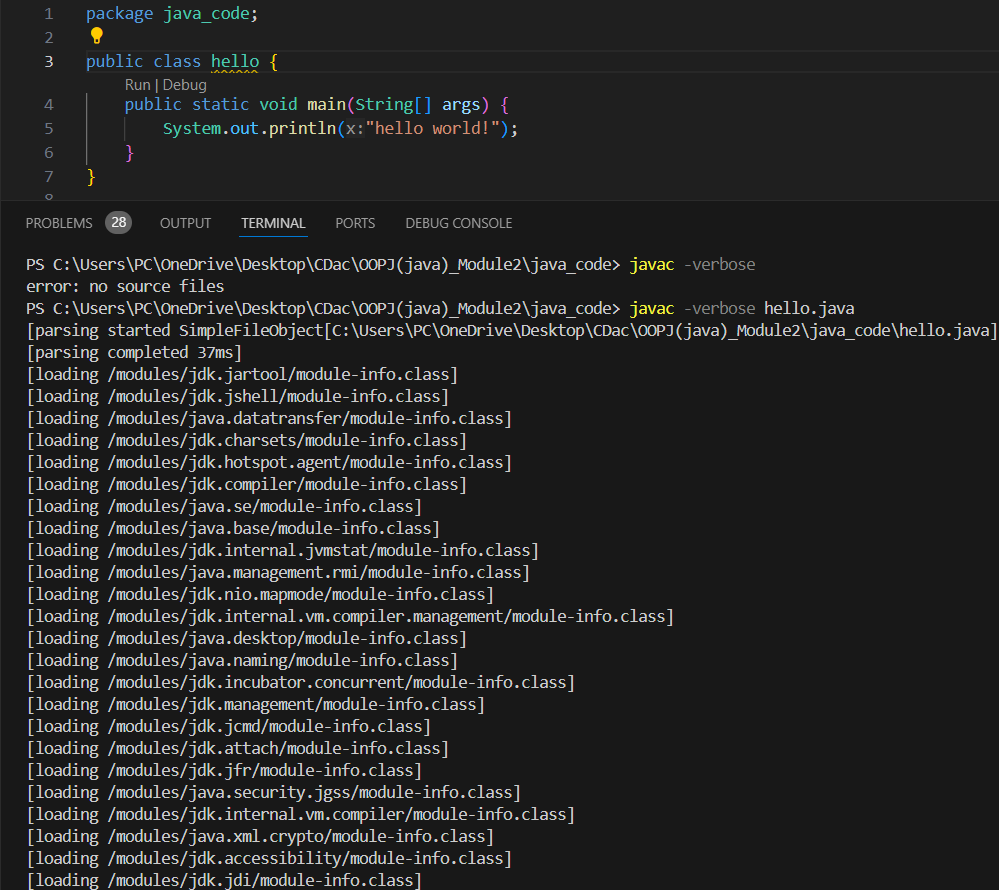
* **Writing Code**: Java programs are written in .java files.
* **Compilation**: The javac compiler converts .java files into bytecode stored in .class files.
* **Execution**: The JVM executes the bytecode, allowing the application to run on any platform that has a compatible JVM.

**6. Coding Assignments**

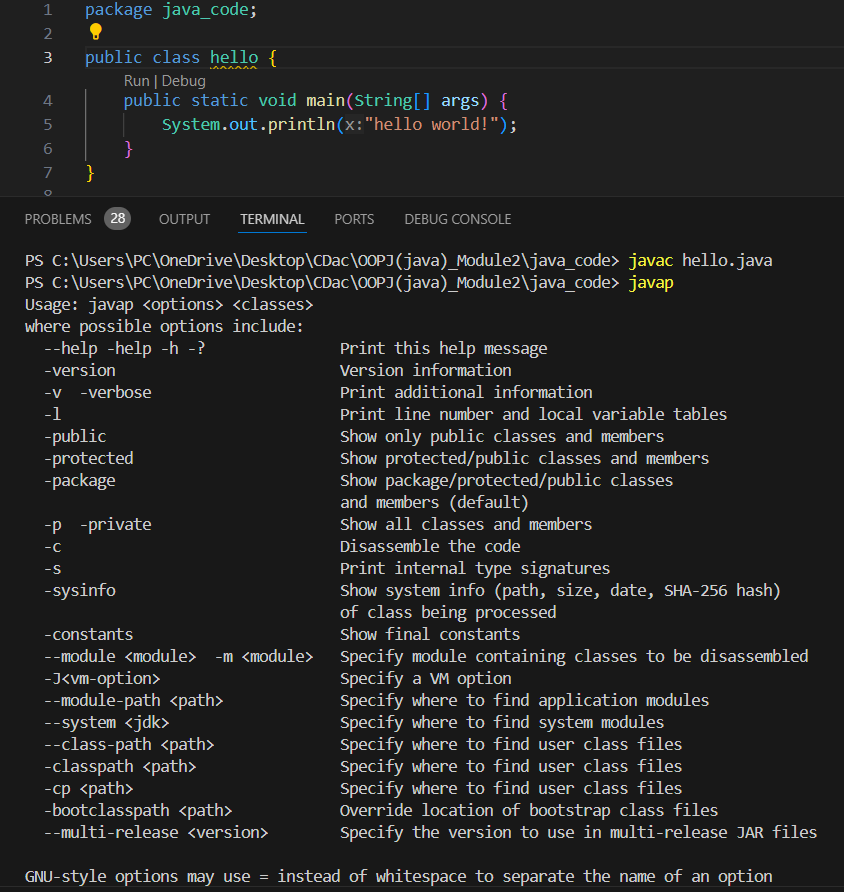
1. **Hello World Program**: Write a Java program that prints "Hello World!!" to the console.



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.



**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>

**JVM Architecture Overview**

\*\*1. **What is the JVM?**

* **Java Virtual Machine (JVM)**: The JVM is an engine that executes Java bytecode, providing a runtime environment for Java applications. It enables Java's platform independence by abstracting the underlying hardware and operating system.

\*\*2. **Key Components**

* **Class Loader**: Loads Java classes into the JVM. It handles class loading, linking, and initialization.
* **Bytecode Interpreter**: Reads and executes Java bytecode instructions. It translates bytecode into machine code for the host system.
* **Just-In-Time (JIT) Compiler**: Compiles bytecode into native machine code at runtime for improved performance.
* **Garbage Collector**: Automatically manages memory by reclaiming memory used by objects that are no longer referenced.

\*\*3. **Memory Areas**

* **Heap**: Stores objects and arrays. Managed by the garbage collector.
* **Stack**: Stores method call frames, local variables, and partial results. Each thread has its own stack.
* **Method Area**: Stores class-level data such as runtime constant pool, field and method data, and method and constructor code.

\*\*4. **Execution Process**

* **Loading**: Class files are loaded into the JVM by the class loader.
* **Verification**: The bytecode is verified for security and correctness.
* **Execution**: The bytecode is either interpreted or compiled to native code by the JIT compiler and executed by the JVM.

**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

**Java Language Environment Overview**

\*\*1. **Core Components**

* **Java Language**: Provides a robust, object-oriented language with features like encapsulation, inheritance, and polymorphism.
* **Java Development Kit (JDK)**: Includes tools and libraries for developing Java applications, such as the compiler (javac), debugger, and runtime libraries.
* **Java Runtime Environment (JRE)**: Provides the necessary runtime environment to execute Java applications, including the Java Virtual Machine (JVM) and core libraries.

\*\*2. **Features**

* **Platform Independence**: Java applications are compiled into bytecode, which can run on any platform with a compatible JVM.
* **Rich API**: Includes libraries for networking, I/O, utilities, and GUI development.
* **Security**: Features like bytecode verification and the security manager enhance the security of Java applications.

\*\*3. **Development Tools**

* **Integrated Development Environments (IDEs)**: Tools like Eclipse and IntelliJ IDEA provide advanced features for coding, debugging, and managing Java projects.