Assignment 1

1. Reading Assignment: A Short History of Java

- **Task**: Read about the history and development of Java.

Java was developed by Sun Microsystems in 1991. The name of the project was Green Project which was lead by Green Team(James Gosling, Patrick Naughton, Mike Sheridan). First the name of java was Oak but due to copywrite issues they change the name to Java. The programming paradigm of Java is object-oriented means everything inside the is within the classes and objects. In 1996 the first version of java "java jdk 1.0" was developed. In 2010 Oracle acquired Sun Microsystem. Java is not only language but also it is a platform. The standardization of java is managed by Java Community Process. Java is object oriented but also supports procedural and functional programming paradigm. The slogan of java is "Write once, run everywhere".

2. Reading Assignment: Java Language Features

- **Simple and Easy to Learn**
  - Java is easy to learn and simple to use as a programming language. It is because of following reasons:
  - Java comprises the same syntax as C, and C++.
  - It holds automatic garbage collection features.
  - Java has its own community to make learning and using Java easy.
- **Object-Oriented Programming**
  - Almost everything written in Java is object and class, making it a true object-oriented programming (OOP) language. The basic concept of OOP is:
  - **Object:** Object is a real-world entity in Java that encompasses state, functionality, and identity.
  - **Class:** Class is a logical entity which includes a group of objects with common properties. It contains fields, methods, constructors, blocks, nested classes and interfaces.
  - **Inheritance:** It's a concept in Java through which developers can create new classes built upon existing classes to achieve runtime polymorphism.
  - **Polymorphism:** A Mechanism in Java through which you can perform a single action in multiple ways. Polymorphism can be of two types- Compile time and runtime.
  - **Abstraction:** It's a method to hide internal processing and show only essential things to the users.
- **Platform Independence**
  - Unlike other languages, Java is not limited to any specific machine and dependent on other factors to run. The Java platform is independent because:
  - It uses a runtime environment of its own, i.e. JVM.
  - It is a write-once, run-anywhere language.

- It is a software-based platform that runs on top of other hardware-based platforms.
- Its code can be executed on multiple platforms, including Windows, Linux, Sun Solaris, and Mac/OS.

Automatic Memory Management

- Automatic memory management is a crucial feature of Java programming. It helps in:
- Automatically allocate and free up space for objects.
- Issues like object destruction don't occur.

Security

- Java programming language is known for its security. With it, you can create virus-free systems because:
- Java programming language runs inside a virtual machine.
- It uses its own runtime environment- JVM.
- Java includes a security manager, which determines what resources a class can access, such as reading and writing to the local disk.
- In Java run time, a class loader separates the package for the classes of the local file system from the files imported from network sources.
- Java also consists of Bytecode Verifier, which checks the code fragments for illegal code.

Rich API

- Being one of the oldest programming languages, Java has a rich API. Some popular Java APIs are:
- Java Advanced Imaging (JAI)
- Java Data Objects (JDO)
- Java Media Frameworks (JMF)
- Java Persistence API (JPA)
- Java 3D (J3D)

Multithreading

- Multithreading is an essential feature of Java that makes Java programming exclusive. It offers several benefits:
- It lets Java developers execute multiple threads at the same time.
- It's used to achieve multitasking.
- It's mostly used in games and animation.

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

It depends on the requirements of a individual. But the most stable version according to the Devs is the jdk 8. Currently I am using jdk 9 which has the following features:

- Platform Module System (Project Jigsaw)
- Interface Private Methods
- Try-With Resources
- Anonymous Classes
- @SafeVarargs Annotation
- Collection Factory Methods
- Process API Improvement
- New Version-String Scheme
- JShell: The Java Shell (REPL)
- Process API Improvement
- Control Panel
- Stream API Improvement
- Installer Enhancement for Microsoft windows and many more

4. Reading Assignment: JDK Installation Directory Structure

- **/jdk-1.8**
- Root directory of the JDK software installation. Contains copyright, license, and README files. Also contains src.zip, the archive of source code for the Java platform.
- **/jdk-1.8/bin**
- Executables for all the development tools contained in the JDK.
  The PATH environment variable should contain an entry for this directory.
- **/jdk-1.8/lib**
- Files used by the development tools. Includes tools.jar, which contains non-core classes for support of the tools and utilities in the JDK. Also includes dt.jar, the DesignTime archive of BeanInfo files that tell interactive development environments (IDEs) how to display the Java components and how to let the developer customize them for an application.
- **/jdk-1.8/jre**
- Root directory of the Java Runtime Environment (JRE) used by the JDK development tools. The runtime environment is an implementation of the Java platform. This is the directory referred to by the java.home system property.
- **/jdk-1.8/jre/bin**
- Executable files for tools and libraries used by the Java platform. The executable files are identical to files in /jdk-1.8/bin. The java launcher tool serves as an application launcher. This directory does not need to be in the PATH environment variable.
- **/jdk-1.8/jre/lib**
- Code libraries, property settings, and resource files used by the JRE. For example rt.jar contains the bootstrap classes, which are the run time classes that

comprise the Java platform core API, and charsets.jar contains the character-conversion classes. Aside from the ext subdirectory, there are several additional resource subdirectories not described here.

- **/jdk-1.8/jre/lib/ext**
- Default installation directory for extensions to the Java platform. This is where the JavaHelp JAR file goes when it is installed, for example. This directory includes the jfxrt.jar file, which contains the JavaFX runtime libraries and the localedata.jar file, which contains the locale data for the java.text and java.util packages. See The Extension Mechanism.
- **/jdk-1.8/jre/lib/security**
- Contains files used for security management. These include the security policy java.policy and security properties java.security files.
- **/jdk-1.8/jre/lib/applet**
- JAR files that contain support classes for applets can be placed in the lib/applet/ directory. This reduces startup time for large applets by allowing applet classes to be preloaded from the local file system by the applet class loader and provides the same protections as though they had been downloaded over the Internet.
- **/jdk-1.8/jre/lib/fonts**
- Font files used by the platform.

6. Coding Assignments

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

```
class Hello{
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

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

```
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
}
```
7. Reading Assignment: The JVM Architecture Explained

The JVM architecture consists of the following components

1. Class Loader Subsystem

- **Purpose:** Loads Java classes into the JVM.
- **Components:**
    - **Bootstrap Class Loader:** Loads essential Java classes needed for the JVM to function.
    - **Platform Class Loader:** Loads the standard Java libraries.
    - **Application Class Loader:** Loads the classes specified in the application's class path.
- **Functionality:** Manages the dynamic loading, linking, and initializing of classes.

2. Runtime Data Areas

The JVM has several memory areas for running Java programs:

- **Method Area:** Stores data about classes, methods, and constants. This area is shared among all threads.
- **Heap:** The memory area where all class instances and arrays are allocated. It's shared across threads and managed by garbage collection.
- **Stack:** Each thread has its own stack for storing local variables, method call details, and partial results. It supports method calls and returns.
- **PC (Program Counter) Register:** Each thread has a PC register that keeps track of the current instruction being executed.
- **Native Method Stack:** Used for executing native methods (written in languages like C or C++). Each thread has its own native stack.

3. Execution Engine

- **Purpose:** Executes the bytecode instructions of Java programs.
- **Components:**
    - **Interpreter:** Reads and executes bytecode instructions one at a time. It's usually slower but flexible.
    - **Just-In-Time (JIT) Compiler:** Compiles bytecode into native machine code while the program is running. This improves performance by reducing the need for interpretation.
    - **Garbage Collector:** Automatically manages memory by cleaning up objects that are no longer needed. It helps in memory management and prevents memory leaks.

4. Native Interface

- **Purpose:** Allows Java code to interact with native applications and libraries.
- **Components:**
    - **Java Native Interface (JNI):** Enables Java code to call native code (like C or C++) and for native code to call Java code.

5. Native Method Libraries

- **Purpose:** Contains the native code libraries used by the JVM.
- **Functionality:** These libraries are usually specific to the platform and provide functionalities not directly available in Java.