**ASSIGNMENT 1**

**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>

The history of Java, as detailed in SunWorld magazine, offers a deep dive into the early development, challenges, and eventual success of the Java programming language. Here's an overview based on the information from SunWorld:

1. The Beginnings: The Green Project

Java's origins trace back to a small team at Sun Microsystems, known as the "Green Team," formed in 1990. The project was initiated by Patrick Naughton, James Gosling, and Mike Sheridan, who were tasked with exploring new ways to program consumer electronics. This team worked on a project called "Oak" (later renamed Java), intending to create a platform-independent language for embedded systems in appliances like set-top boxes.

2. \**Oak and the 7 Prototype*

The first significant development from the Green Team was Oak, designed to be a robust, platform-independent language. They built a prototype device called \*7, which was an interactive, handheld home entertainment controller. The goal was to make Oak the language that would run on these devices.

3. FirstPerson Inc. and the Set-Top Box Market

In 1993, Sun Microsystems formed a subsidiary named FirstPerson Inc. to commercialize the technology developed by the Green Team. They focused on the emerging set-top box market and bid on a major project from Time-Warner to create an operating system for set-top boxes and video-on-demand technology. Despite having a superior technical solution, they lost the bid due to business politics.

4. Shift in Focus: The Collapse of FirstPerson

After losing the Time-Warner bid and realizing that the market for interactive TV wasn't ready, the Green Team shifted its focus. In early 1994, FirstPerson was dissolved, and some of its members moved to Sun Interactive to work on digital video servers. The remaining team members began focusing on applying their technology to network-based desktop systems, marking a significant pivot from their original consumer electronics vision.

5. The Birth of Java

As the team shifted away from set-top boxes, the focus turned towards creating a language that could run on any device, from computers to handheld devices. The language was renamed "Java" after a legal issue with the name "Oak." Java was designed to be platform-independent, object-oriented, and secure, making it ideal for networked environments like the burgeoning World Wide Web.

6. Java Takes Off: 1995 and Beyond

Java was officially launched in 1995, and it quickly became a cornerstone of web development. Sun Microsystems marketed Java as "write once, run anywhere," emphasizing its ability to run on any device with a compatible Java Virtual Machine (JVM). The language gained massive popularity for its portability and robustness, becoming a key tool in web development, enterprise applications, and mobile platforms.

7. Java’s Impact

Java's introduction revolutionized programming by allowing developers to create applications that could run on multiple platforms without needing to be rewritten. It became the backbone for many web services, mobile applications, and enterprise software, solidifying its place as one of the most important programming languages in the world.

8. Java in the 21st Century

Java continued to evolve with the growth of the internet and mobile technologies. It became the language of choice for Android app development and continued to be widely used in enterprise environments. Over the years, Java has undergone several updates and revisions, ensuring it remains relevant in the ever-changing technology landscape.

Conclusion

Java's journey from a project aimed at consumer electronics to becoming one of the most influential programming languages is a testament to the vision and adaptability of its creators. Despite early setbacks, the strategic pivots made by the Green Team led to the development of a language that would have a lasting impact on the world of technology.

This history, as documented by SunWorld, highlights the innovation, challenges, and strategic decisions that shaped Java into the powerhouse it is today.

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

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

The [Java Almanac](https://javaalmanac.io/features/) provides an overview of Java language features and enhancements across different versions. Here’s a summary of some notable Java language features from various versions that you might find on such a site:

**Java Language Features**

1. **Java 1.0 (1996)**
   * **Core Features**: Object-oriented programming, platform independence, basic class libraries, and the Java Virtual Machine (JVM).
2. **Java 1.1 (1997)**
   * **New Features**: Inner classes, JDBC (Java Database Connectivity), RMI (Remote Method Invocation), and reflection.
3. **Java 1.2 (1998)**
   * **Major Enhancements**: Collections framework, Java Foundation Classes (Swing), and the introduction of the java.lang package improvements.
4. **Java 1.3 (2000)**
   * **Improvements**: HotSpot JVM improvements, J2EE (Java 2 Platform, Enterprise Edition) enhancements, and new I/O classes.
5. **Java 1.4 (2002)**
   * **New Features**: Assertions, exception chaining, NIO (New I/O), logging API, and regular expressions.
6. **Java 5 (2004)**
   * **Significant Updates**: Generics, metadata annotations, enumerated types, enhanced for loop, and autoboxing/unboxing.
7. **Java 6 (2006)**
   * **Enhancements**: Scripting API (support for JavaScript through JSR 223), improvements in JVM performance, and new I/O libraries.
8. **Java 7 (2011)**
   * **Key Features**: Try-with-resources statement, diamond operator, string in switch statements, and fork/join framework for parallel processing.
9. **Java 8 (2014)**
   * **Major Updates**: Lambda expressions, functional interfaces, streams API, and new Date and Time API (java.time).
10. **Java 9 (2017)**
    * **New Features**: Module system (Project Jigsaw), JShell (REPL), and improved Javadoc with search capabilities.
11. **Java 10 (2018)**
    * **Enhancements**: Local-variable type inference (var), application class-data sharing, and other performance improvements.
12. **Java 11 (2018)**
    * **Key Updates**: Long-Term-Support (LTS) release, new HTTP client API, and removal of deprecated features.
13. **Java 12 (2019)**
    * **New Features**: Shenandoah garbage collector (experimental), microbenchmark suite, and various performance improvements.
14. **Java 13 (2019)**
    * **Enhancements**: Text blocks (preview feature), dynamic CDS (Class Data Sharing) archives, and improvements to the JVM.
15. **Java 14 (2020)**
    * **New Features**: Records (preview feature), pattern matching for instanceof (preview feature), and the new Garbage Collector (ZGC).
16. **Java 15 (2020)**
    * **Key Updates**: Sealed classes (preview feature), hidden classes, and improvements to the string handling API.
17. **Java 16 (2021)**
    * **Enhancements**: Finalized records and pattern matching for instanceof, new vector API (incubator), and improved JVM performance.
18. **Java 17 (2021)**
    * **LTS Release**: Sealed classes, pattern matching for switch (preview), and strong encapsulation of JDK internals.
19. **Java 18 (2022)**
    * **New Features**: UTF-8 by default, and improvements to the standard library and tooling.
20. **Java 19 (2022)**
    * **Enhancements**: Record patterns (preview), and virtual threads (preview).
21. **Java 20 (2023)**
    * **New Features**: More previews and incubator features, including continued development on virtual threads and record patterns.
22. **Java 21 (2023)**
    * **LTS Release**: Pattern Matching, Virtual Threads, and further improvements and optimizations.

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

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

**Recommended JDK Versions**

1. **JDK 21 (LTS)**
   * **Release Date**: September 2023
   * **Features**: Pattern Matching, Virtual Threads
   * **Recommendation**: Current Long-Term-Support (LTS) version. Suitable for most use cases, especially for production environments requiring stability and extended support.
2. **JDK 17 (LTS)**
   * **Release Date**: September 2021
   * **Features**: Sealed Classes
   * **Recommendation**: Supported LTS version. Consider upgrading to JDK 21 for newer features and improvements, but JDK 17 remains a good option if you're already using it.

**Choosing a Distribution**

1. **Adoptium Eclipse Temurin**
   * **Recommendation**: Highly recommended for its high quality, vendor-neutral approach, and regular updates. It’s a good choice for most use cases.
2. **Azul Zulu**
   * **Recommendation**: A good choice, offering no-cost, production-ready builds. It supports a wide range of platforms and provides stabilized security updates.
3. **BellSoft Liberica JDK**
   * **Recommendation**: A solid choice with good industry reputation. It’s compatible with various operating systems and architectures.
4. **Amazon Corretto**
   * **Recommendation**: A good option, particularly if you are running Java applications on AWS infrastructure.
5. **Microsoft Build of OpenJDK**
   * **Recommendation**: Use primarily if you are running Java applications directly on Azure.

**Avoid**

1. **OpenJDK builds by Oracle**
   * **Reason**: Limited update period and licensing issues.
2. **Oracle Java SE Development Kit (JDK)**
   * **Reason**: Licensing complexity and potential costs for production use.
3. **AdoptOpenJDK**
   * **Reason**: It has been succeeded by Adoptium Eclipse Temurin.
4. **Alibaba Dragonwell, SapMachine, Red Hat OpenJDK**
   * **Reason**: Generally less recommended unless specific to your infrastructure needs.
5. **ojdkbuild**
   * **Reason**: Project is discontinued.
6. **GraalVM**
   * **Reason**: While powerful, its suitability for production depends on specific needs and use cases.

**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>

**/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](https://docs.oracle.com/javase/8/docs/technotes/guides/extensions/index.html).

**/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.

**Additional Files and Directories**

This section describes the directory structure for Java source code, C header files, and other additional directories and files.

jdk-1.8

include

man

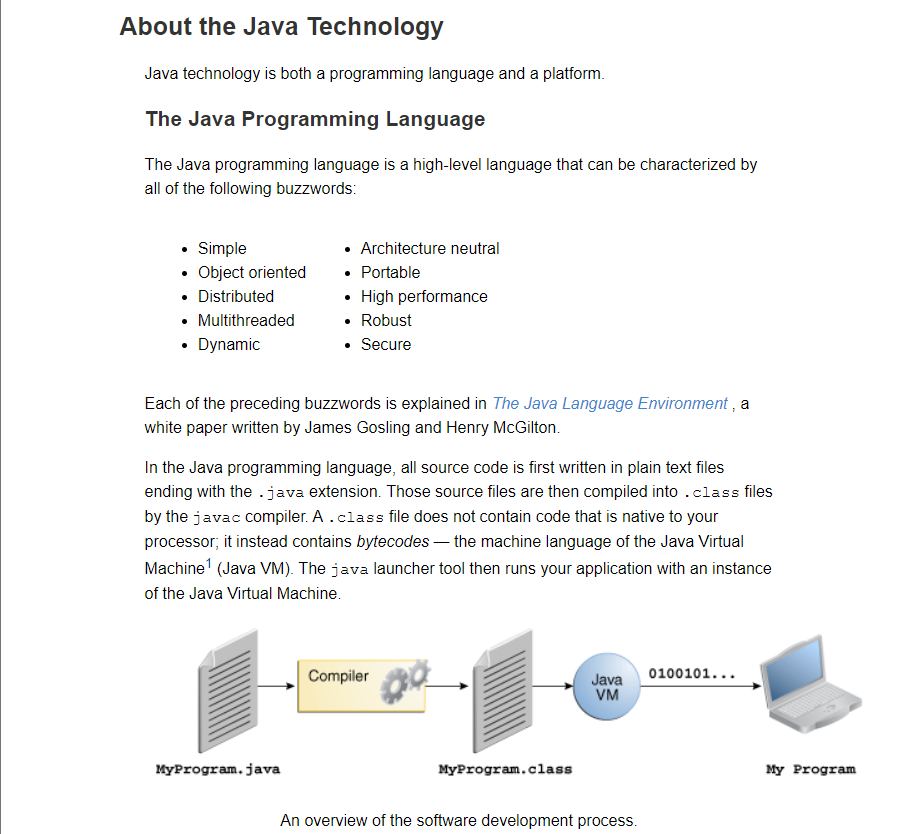
src.zip

**/jdk-1.8/src.zip**

Archive that contains the source code for the Java platform.

**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>

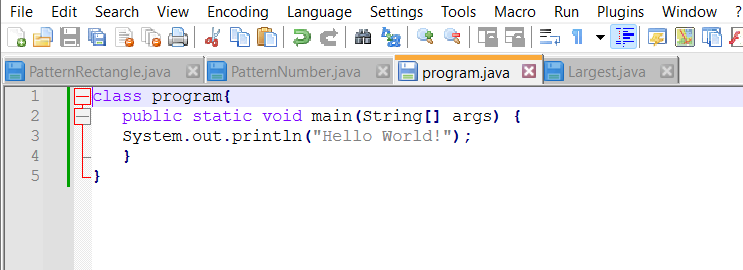


The general-purpose, high-level Java programming language is a powerful software platform. Every full implementation of the Java platform gives you the following features:

* **Development Tools**: The development tools provide everything you'll need for compiling, running, monitoring, debugging, and documenting your applications. As a new developer, the main tools you'll be using are the javac compiler, the java launcher, and the javadoc documentation tool.
* **Application Programming Interface (API)**: The API provides the core functionality of the Java programming language. It offers a wide array of useful classes ready for use in your own applications. It spans everything from basic objects, to networking and security, to XML generation and database access, and more. The core API is very large; to get an overview of what it contains, consult the [Java Platform Standard Edition 8 Documentation](https://docs.oracle.com/javase/8/docs/index.html).
* **Deployment Technologies**: The JDK software provides standard mechanisms such as the Java Web Start software and Java Plug-In software for deploying your applications to end users.
* **User Interface Toolkits**: The JavaFX, Swing, and Java 2D toolkits make it possible to create sophisticated Graphical User Interfaces (GUIs).
* **Integration Libraries**: Integration libraries such as the Java IDL API, JDBC API, Java Naming and Directory Interface (JNDI) API, Java RMI, and Java Remote Method Invocation over Internet Inter-ORB Protocol Technology (Java RMI-IIOP Technology) enable database access and manipulation of remote objects.

**6. Coding Assignments**

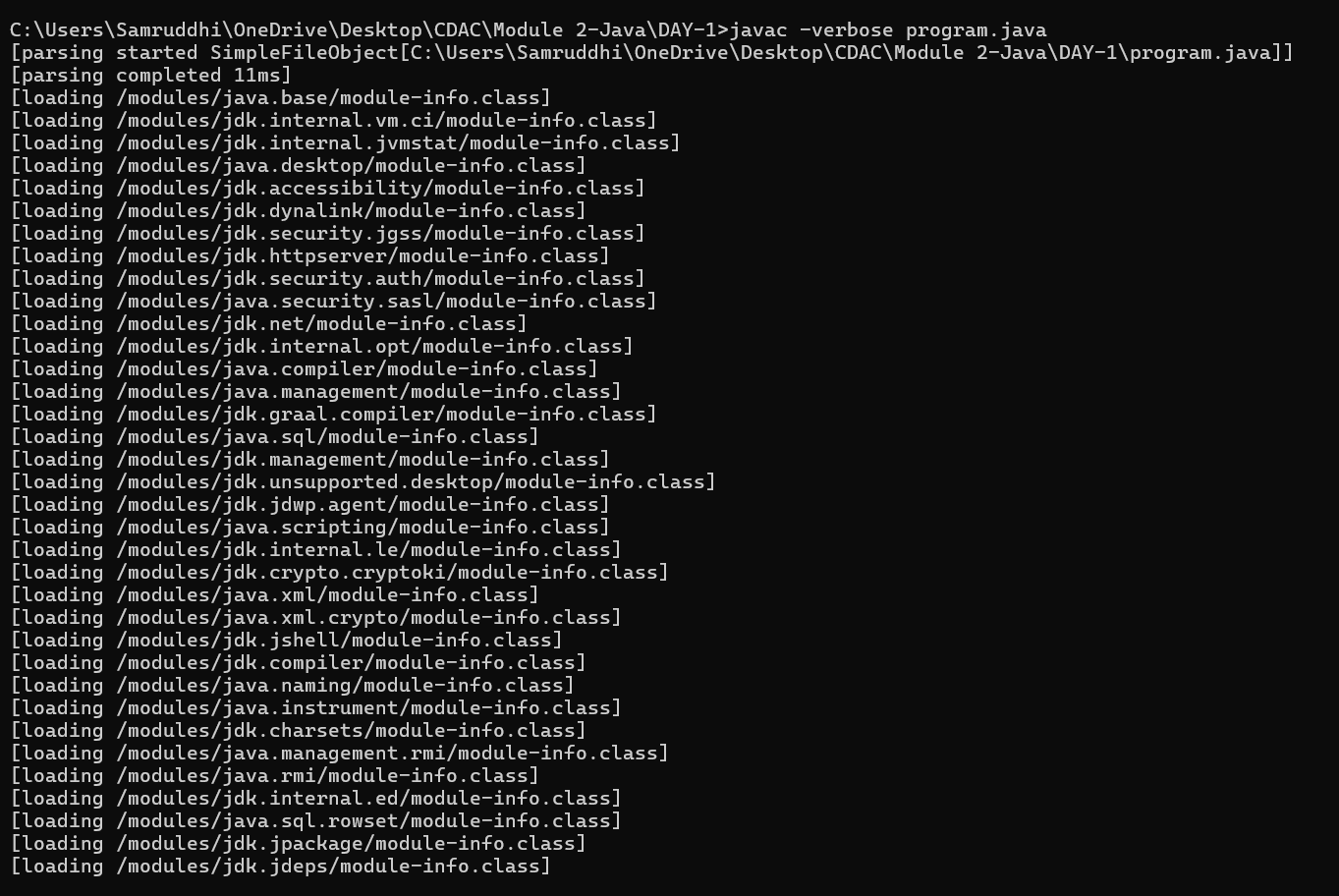
1. **Hello World Program**: Write a Java program that prints "Hello World!!" to the console.



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

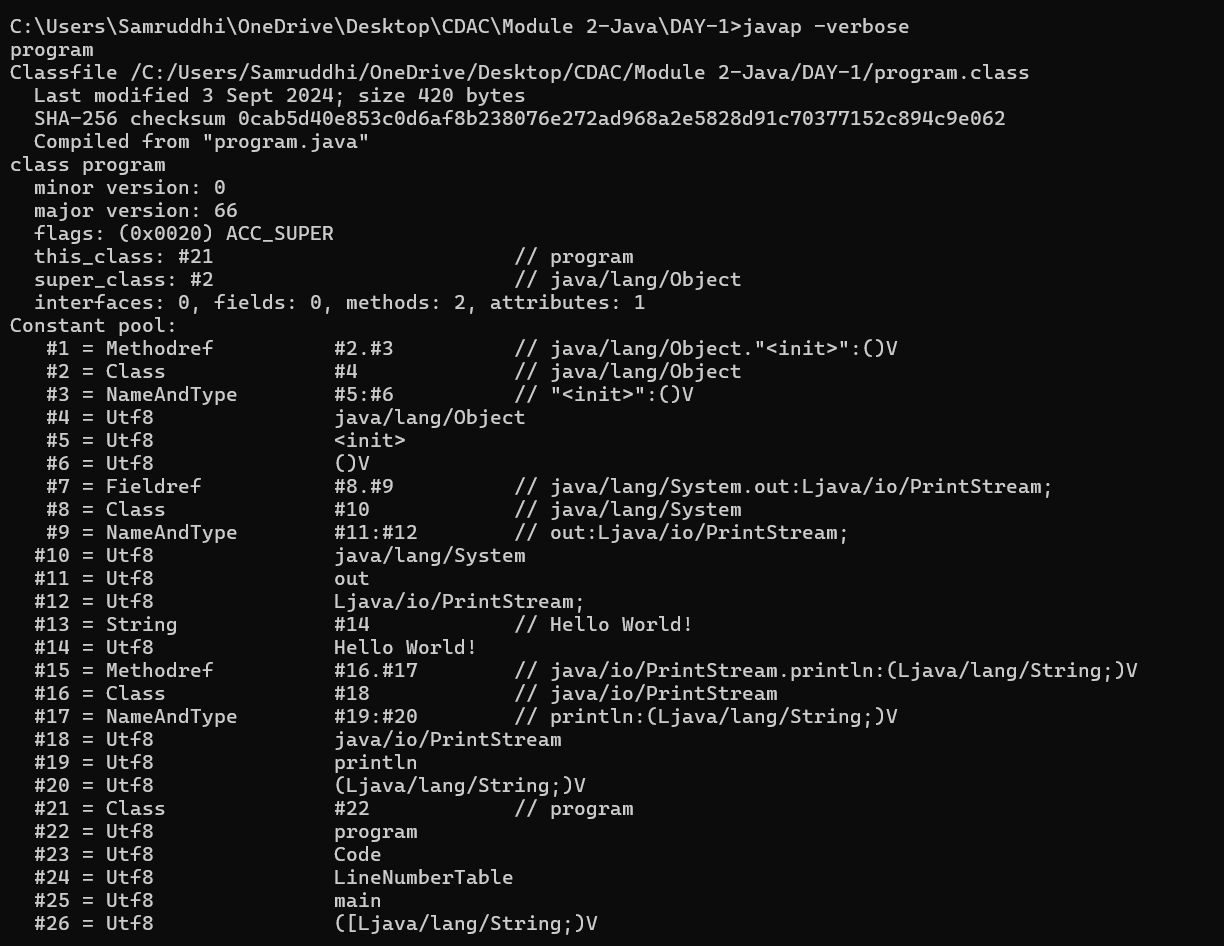
When you compile with the -verbose option, javac will provide detailed information about the compilation process, including:

* **File Reading**: It will show messages about reading the source file.
* **Loading**: Information about loading classes and packages.
* **Compiling**: Details on the compilation stages, such as parsing, generating code, and writing output files.
* **Other Details**: Any additional diagnostic information that can help in understanding what the compiler is doing behind the scenes.



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

**Common Options for javap**

****

* **-c**: Disassembles the bytecode into human-readable form.

javap -c Program

* **-p**: Shows private members in addition to public and protected ones.

javap -p Program

* **-verbose**: Provides a more detailed output, including additional information about constants and class attributes.

javap -verbose Program

* **-s**: Shows internal type signatures of fields and methods.

javap -s Program

**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>

**How Does the JVM Work?**

As shown in the above architecture diagram, the JVM is divided into three main subsystems:

1. ClassLoader Subsystem
2. Runtime Data Area
3. Execution Engine

**1. ClassLoader Subsystem**

Java's [dynamic class loading](http://www.javainterviewpoint.com/use-of-class-forname-in-java/) functionality is handled by the ClassLoader subsystem. It loads, links. and initializes the class file when it refers to a class for the first time at runtime, not compile time.

**1.1 Loading**

Classes will be loaded by this component. BootStrap ClassLoader, Extension ClassLoader, and Application ClassLoader are the three ClassLoaders that will help in achieving it.

1. **BootStrap [ClassLoader](http://www.javainterviewpoint.com/" \t "_blank)** – Responsible for loading classes from the bootstrap classpath, nothing but **rt.jar.**Highest priority will be given to this loader.
2. **Extension ClassLoader** – Responsible for loading classes which are inside the ext folder **(jre\lib).**
3. **Application ClassLoader** –Responsible for loading Application Level Classpath, path mentioned Environment Variable, etc.

The above ClassLoaders will follow Delegation Hierarchy Algorithm while loading the class files.

**1.2 Linking**

1. **Verify** – Bytecode verifier will verify whether the generated bytecode is proper or not if verification fails we will get the verification error.
2. **Prepare** – For all static variables memory will be allocated and assigned with default values.
3. **Resolve** – All symbolic memory references are replaced with the original references from Method Area.

**1.3 Initialization**

This is the final phase of ClassLoading; here, all [static variables](http://www.javainterviewpoint.com/use-of-static-keyword-in-java/) will be assigned with the original values, and the [static block](http://www.javainterviewpoint.com/java-static-import/) will be executed.

**2. Runtime Data Area**

The Runtime Data Area is divided into five major components:

1. **Method Area** – All the class-level data will be stored here, including static variables. There is only one method area per JVM, and it is a shared resource.
2. **Heap Area** – All the Objects and their corresponding instance variables and arrays will be stored here. There is also one Heap Area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread-safe.
3. **Stack Area**– For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called Stack Frame. All local variables will be created in the stack memory. The stack area is thread-safe since it is not a shared resource.
4. **PC Registers** – Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.
5. **Native Method stacks** – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.

**3. Execution Engine**

The bytecode, which is assigned to the **Runtime Data Area,** will be executed by the Execution Engine. The Execution Engine reads the bytecode and executes it piece by piece.

1. **Interpreter** – The interpreter interprets the bytecode faster but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.
2. **JIT Compiler**– The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.
3. **Garbage Collector**: Collects and removes unreferenced objects. Garbage Collection can be triggered by calling System.gc(), but the execution is not guaranteed. Garbage collection of the JVM collects the objects that are created.

**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 Specification**:

* **Description**: Details the syntax and semantics of the Java programming language. It defines how Java code should be written and how it should behave during execution.

 **Java SE (Standard Edition)**:

* **Description**: Provides core functionality for Java applications. It includes the Java Runtime Environment (JRE) and the Java Development Kit (JDK). Java SE is designed for developing and running desktop and server applications.

 **Java EE (Enterprise Edition)**:

* **Description**: Now known as Jakarta EE, it extends Java SE with additional libraries and APIs for enterprise applications, including web services, messaging, and more. It's used for building large-scale, distributed, and multi-tiered applications.

 **Java ME (Micro Edition)**:

* **Description**: A subset of Java SE designed for mobile and embedded devices. It provides a smaller footprint suitable for devices with limited resources.

 **Java FX**:

* **Description**: A framework for building rich internet applications with a modern user interface. It provides a more sophisticated UI toolkit compared to Swing.

 **Java Virtual Machine (JVM)**:

* **Description**: The JVM executes Java bytecode and provides a platform-independent environment. It handles memory management, garbage collection, and thread management.

 **Java Development Kit (JDK)**:

* **Description**: Contains tools for developing Java applications, including the Java compiler (javac), the Java Runtime Environment (JRE), and various development tools and libraries.

 **Java Runtime Environment (JRE)**:

* **Description**: Provides the libraries, Java Virtual Machine (JVM), and other components necessary to run Java applications.

 **Java Community Process (JCP)**:

* **Description**: The process by which the Java platform evolves. It involves various stakeholders and organizations to develop and review Java technology specifications.