Diagram

Description automatically generated

Java is a high level, platform-independent and object-oriented programming language created in 1995 by Oracle.

*Before deep-diving into Java, we must understand the differences between JDK, JRE and JVM.*

**JDK** :- Java Development Kit(JDK) is a software development kit that includes development tools. It provides an environment to develop and execute the java code.

**JRE** :- Java Runtime Environment is the **implementation of JVM.**It provides a runtime environment to the java program.

**JVM** :- Java Virtual Machine is a physically non-existing abstract machine. Therefore, it can neither be installed nor uninstalled like other software. JVM is a **specification** that specifies how things should be done.

**Types of Virtual Machines**

There are two types of virtual machines.

1. System-based Virtual Machines
2. Application-based Virtual Machines (Process-based VMs)

Table

Description automatically generated with medium confidence

Types of VMs and their differences

Furthermore, executing a java program will create a JVM instance on your computer.

Example: When there are 2 java programs executing same time, there are 2 JVM instances.

When there is a JVM instance, a [non-daemon thread](https://www.geeksforgeeks.org/difference-between-daemon-threads-and-user-threads-in-java/#:~:text=Daemon%20threads%20are%20low%20priority,used%20to%20perform%20supporting%20tasks.) is also there. There are two ways that JVM instances get destroyed.

1. When there is no non-daemon thread exists.
2. When the program calls system.exit() method.

Therefore, JVM only exists when the application runs or at least one non-daemon thread exists.

**JVM Components**

Imagine a kid who wants to throw balls to a net. First, he needs to take some balls from a lot, then stores them in a basket, and finally, he can throw balls to the net. When we apply this scenario to JVM, JVM first loads classes into the area stores them in the memory area, and finally executes them through the execution engine.

*There are three components of JVM. Those are,*

1. ClassLoader
2. Memory Area
3. Execution Engine

*Now let’s focus on structure of JVM*

Diagram

Description automatically generated

Structure of JVM

**ClassLoader**

The main responsibility of the classloader is loading class information to the memory area. There are two types of classloaders.

1. Bootstrap Classloader
2. Custom Classloader

Classloader has **three stages**, those are **Loading, Linking, Initialization.**

1. **Loading**

Before loading the class, JVM reads several things in the loading stage.

* Reads fully qualified class name.
* Reads immediate parent information.
* Reads variable information.
* Reads whether it is **a Class, an Interface or an enum**.

After JVM reads the above facts, the Class file loads to the memory area. Also, when loading a class, JVM creates class type object and stores it on the heap for the first time.

**2. Linking**

The linking again can be divided into three stages —**Verification, Preparation, Resolution.**

* Verification — In here a sub-program called **“bytecode verifier”** checks whether the class came from **valid compiler**, whether it has **correct format** and whether is has **correct structure**. To complete the verification stage successfully, need to fulfil all the above mentioned conditions, otherwise it throws “VerifierException”.
* Preparation — In this stage, if we use any static or instance variable in our class it will assign a default value.
* Resolution — In java is allows us to use domain specific objects(business objects) on our program, but before it comes to machine level JVM replaces symbolic links with direct links.

**3. Initialization**

In this stage, it assigns real values, and also it execute static blocks. **Each class must be initialize before active use.**

Here are the things that consider active use of the class. Before any below mentioned active use, it must go through the initialiation.

* “new” keyword.
* Invocation of static method.
* Using Reflection API to load the class(getInstance).
* Assigning values to static field.
* If class is initial (main method).
* Instantiate sub class.

There are four ways to initialize a class. Those are use “new” keyword, clone() method, Reflection API and IO.ObjectInputStram class.

**Memory Area**

1. Method Area — The area that holds class information.
2. Heap — The area that holds object information.
3. Stack — The area that holds local variable information.
4. PC Registers — The area that holds information about the next execution (if it is not a native method)
5. Native Method Area — The area that holds native method and method information.

**Execution Engine**

The execution engine consist of three main components. **Interpreter, JITcompiler and Garbage Collector** are them. Execution engine fulfil the communication process between JVM and memory areas. Also the execution of the bytecode is happens on here.

* Interpreter — Interpreter reads the bytecode and interpret it to machine code.
* JIT Compiler — When interpreter interpret bytecode to machine code, it interprets the same method every time, this redices the system performance. JIT Compiler is there to solve this issue. It compiles bytecode into native machine code at runtime.
* Garbage Collector — Garbage Collector is a useful program when it comes to memory management. It destroys the unreachable methods in heap and recycle the memory areas.