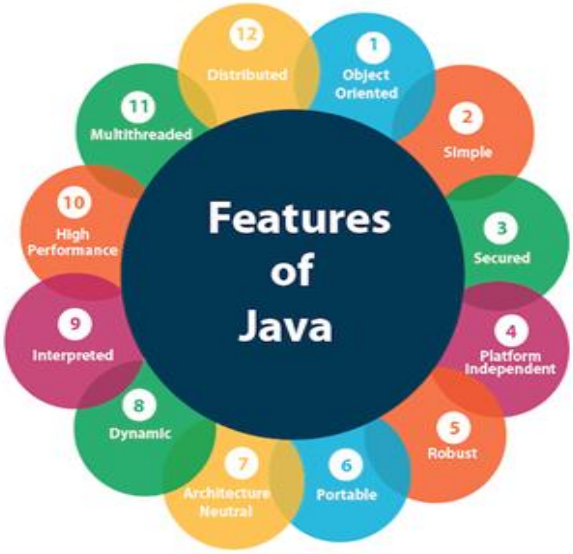
1.1 OVERVIEW OF JAVA PLATFORM AND FEATURES, JVM, JRE, JDK

- Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995.
- *James Gosling* is known as the father of Java.
- Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.
- Any hardware or software environment in which a program runs is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

1. Object-oriented:



- Java is an object-oriented programming language. Everything in Java is an object.
- Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

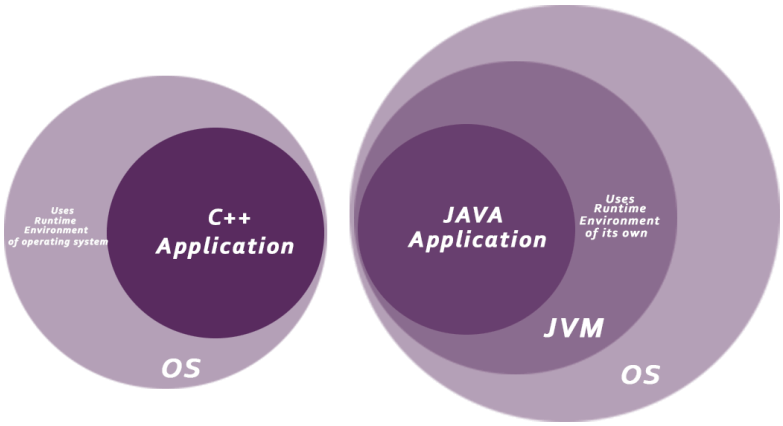
2. Simple:

- Java is very easy to learn, and its syntax is simple, clean and easy to understand.
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

3. **Secure:**

Java program cannot harm another system thus making it safe. Java is secured because:

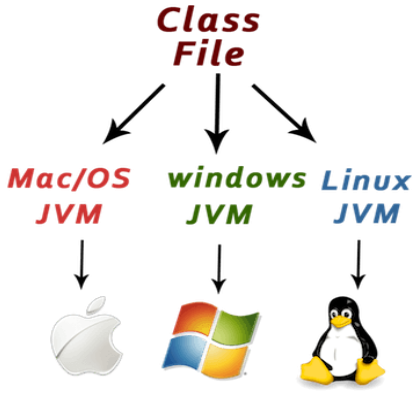
- No explicit pointer.
- Java Programs run inside virtual machine sandbox.



4. **Platform Independent:**

➢ When we write Java code, it is first compiled by the compiler and then converted into bytecode (which is platform-independent).

➢ This byte code can run on any platform which has JVM installed.



5. **Robust:**

Java is robust because:

➢ It uses strong memory management.

➢ There is a lack of pointers that avoids security problems.

➢ There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.

➢ There are exception handling and the type checking mechanism in Java. All these points make Java robust.

6. **Portable:**

➢ Java programs can execute in any environment for which there is a Java run-time system (JVM).

➢ Java programs OR codes can be run on any platform means OS (Linux, Window, Mac). Java programs can be shifted over World Wide Web (e.g., applets)

➢ We may carry the java byte code to any platform.

7. **Architecture Neutral:**

- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

8. **Dynamic:**

- Java is a dynamic language.
- It supports dynamic loading of classes. It means classes are loaded on demand.
- It also supports functions from its native languages, i.e., C and C++.
- Java supports dynamic compilation and automatic memory management (garbage collection).

9. **Interpreted:**

- Java code is not directly executed by the computer.
- It is first compiled into bytecode.
- This byte code is then understand by the JVM. This enables Java to run on any platform without rewriting code.

10. **High Performance:**

- Java is faster than old interpreted languages.
- Java program is first converted into bytecode which is faster than interpreted code.
- It is slower than fully compiled languages like C or C++ because of interpretation and JIT compilation process.
- Java performance is improve with the help of Just-In-Time (JIT) compilation, which makes it faster than many interpreted languages but not as fast as fully compiled languages.

11. **Multi-Threaded:**

Multithreading in Java allows multiple threads to run at the same time.

- Java provides build in support for managing multiple threads. A **thread** is known as the smallest unit of execution within a process.
- It improves CPU utilization and enhancing performance in applications that require concurrent task execution.
- It is especially important for interactive and high-performance applications, such as games and real-time systems.

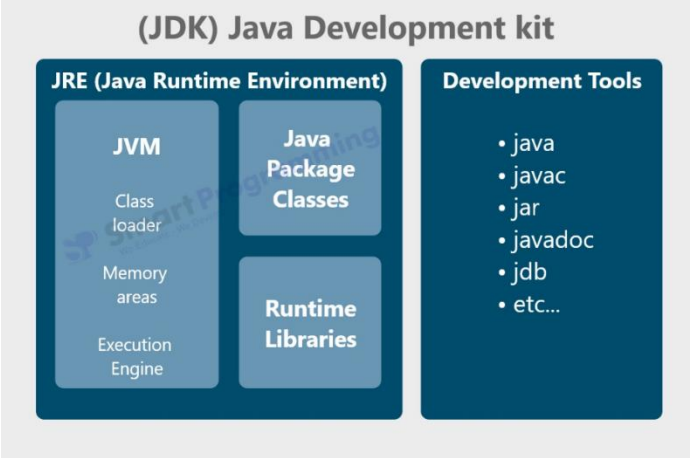
12. **Distributed:**

- Java is distributed because it facilitates users to create distributed applications in Java.
- RMI and EJB are used for creating distributed applications.
- This feature of Java makes us able to access files by calling the methods from any machine on the internet.

JDK - Java Development Kit:

- It is a software development environment which is used to develop Java applications.
- It physically exists and contains JRE along with development tools.
- JDK is an implementation of any one of the below java platforms released by Oracle Corporation, which are as follows.

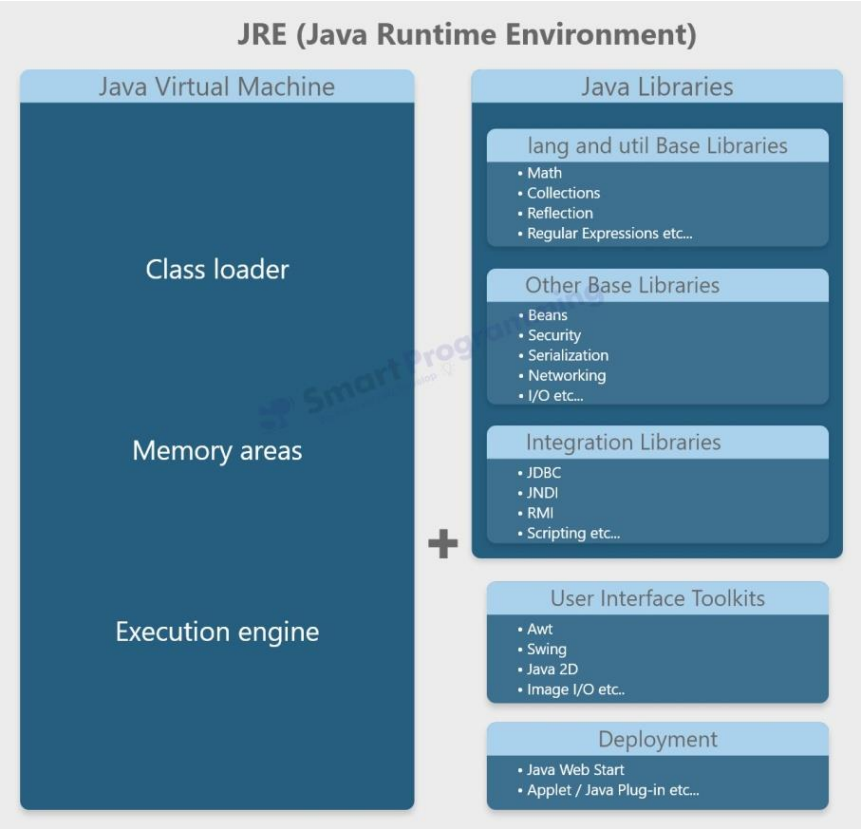
❖ Standard Edition Java Platform
❖ Enterprise Edition Java Platform
❖ Micro Edition Java Platform



- In general, JDK contains a private Java Virtual Machine (JVM) and a few other resources like interpreter/loader, java compiler, archivers (jar files) and Javadoc for the documentation generator to complete the development of Java Applications.

JRE - Java Runtime Environment:

- The JRE is a software layer that runs on top of a computer's operating system software and provides the class libraries and other required resources to run a specific java program.
- JRE is one of the three interrelated components for developing Java programs along with JDK and JVM.
- The JDK and JRE are the two components that interact with each other to create a sustainable run-time environment that enables the seamless execution of Java-based applications in virtually any operating system**.**
- JRE run-time architecture generally uses the ClassLoader, Bytecode verifier, and Interpreter.
- In which ClassLoader dynamically load all the necessary classes to run a Java program. It also automates the loading of classes as per requirement, so it can also effectively utilize the memory.

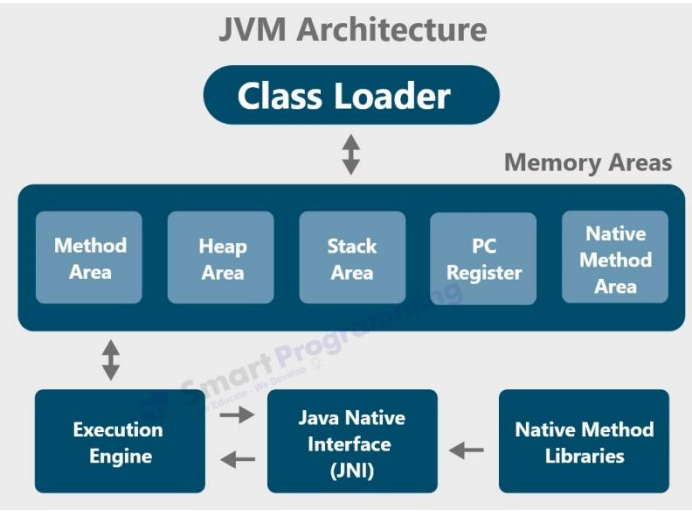


- Bytecode verifier checks the format and accuracy of Java code before it passes to the interpreter.
- And finally, the interpreter helps to run the java program natively on the underlying machine.

JIT – Just In Time & JVM - Java Virtual Machine:

There are 3 main components in JVM :-

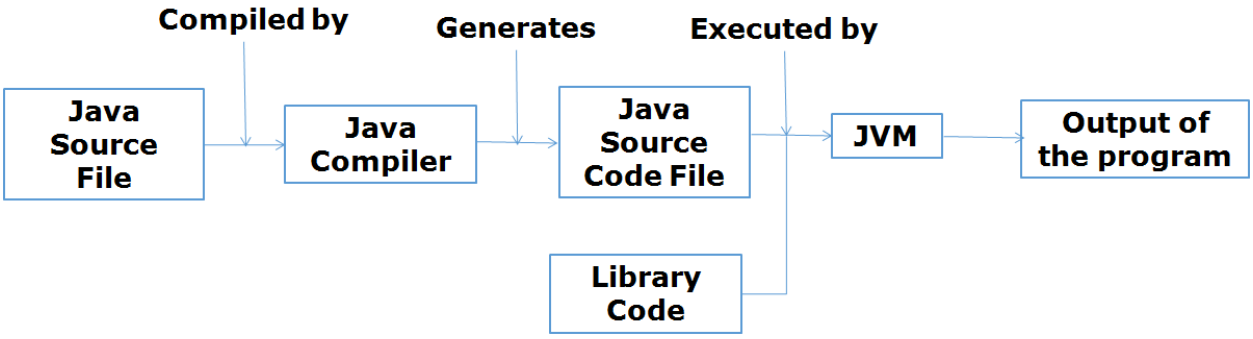
1. **Class Loader:** The class loader is responsible for dynamically loading Java classes into memory at runtime. It handles loading, linking, and initialization, organizing classes into namespaces and ensuring they are available for execution.



2. **Memory Areas:** The JVM's memory areas include the heap (for objects), stack (for method calls and local variables), method area (for class-level data), and native method stacks. These regions manage memory allocation, storage of program data, and execution context.

3. **Execution Engine:** The execution engine interprets or compiles byte code into machine code for execution by the CPU. It includes the interpreter for line-by-line execution and the Just-In-Time (JIT) compiler for optimizing frequently executed code paths.

- JIT is an integral part of JVM and Garbage Collector.
- It is capable of compiling the Java code into machine code, which can significantly improve the performance of Java Applications.
- While the compilation of the codes, it is not guaranteed which code will be compiled. JIT usually compile the code as per its requirement to JVM.
- Once method execution crosses the certain limit (of approx. 10K execution), then it will be converted into machine code.
- To run a simple program in Java programming, we have to use the Java compiler to convert the source code into Java class code. After that, JVM is responsible for running the program.



1.2. STRUCTURE OF JAVA PROGRAM COMPILATION AND RUNNING

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

For the above program file name should be**: Simple.java**

(Note: Main class name should be the file name of every java program)

To compile:

```
javac Simple.java
```

To execute:

java Simple

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** is used for command line argument.
- **System.out.println()** is used print statement.

A program is written in JAVA, the **javac** compiles it. The result of the JAVA compiler is the **.class** file or the **bytecode** and not the machine native code (unlike C compiler). The bytecode generated is a non-executable code and needs an interpreter to execute on a machine. This interpreter is the JVM and thus the Bytecode is executed by the JVM. And finally program runs to give the desired output.