Module 1:

Java:

- Java is a widely-used programming language for coding web applications.
- Java is a multi-platform, object-oriented, and network-centric language that can be used as a platform in itself.
- It is a fast, secure, reliable programming language for coding everything from mobile apps and enterprise software to big data applications and server-side technologies.

Features Of Java:

Simple:

- Java is a simple programming language and easy to understand because it does not contain complexities that exist in prior programming languages.
- Java contains the same syntax as C, and C++, so the programmers who are switching to Java will not face any problems in terms of syntax.
- Secondly, the concept of pointers has been completely removed from Java which leads to confusion for a programmer and pointers are also vulnerable to security.

Object Oriented:

- Java is an Object-Oriented Programming Language, which means in Java everything is written in terms of classes and objects.
- The main concepts of any Object-Oriented Programming language are given below:

1. Class and Object
2. Encapsulation
3. Abstraction
4. Inheritance
5. Polymorphism

Platform Independent:

- Java allows programmers to write their program on any machine with any configuration and to execute it on any other machine having different configurations.
- Java source code is compiled to bytecode and this bytecode is not bound to any platform. In fact, this bytecode is only understandable by the Java Virtual Machine which is installed in our system.
- Every operating system has its own version of JVM, which is capable of reading and converting bytecode to an equivalent machine’s native language.

Portable:

- The WORA (Write Once Run Anywhere) concept and platform-independent feature make Java portable. Now using the Java programming language, developers can yield the same result on any machine, by writing code only once. The reason behind this is JVM and bytecode.

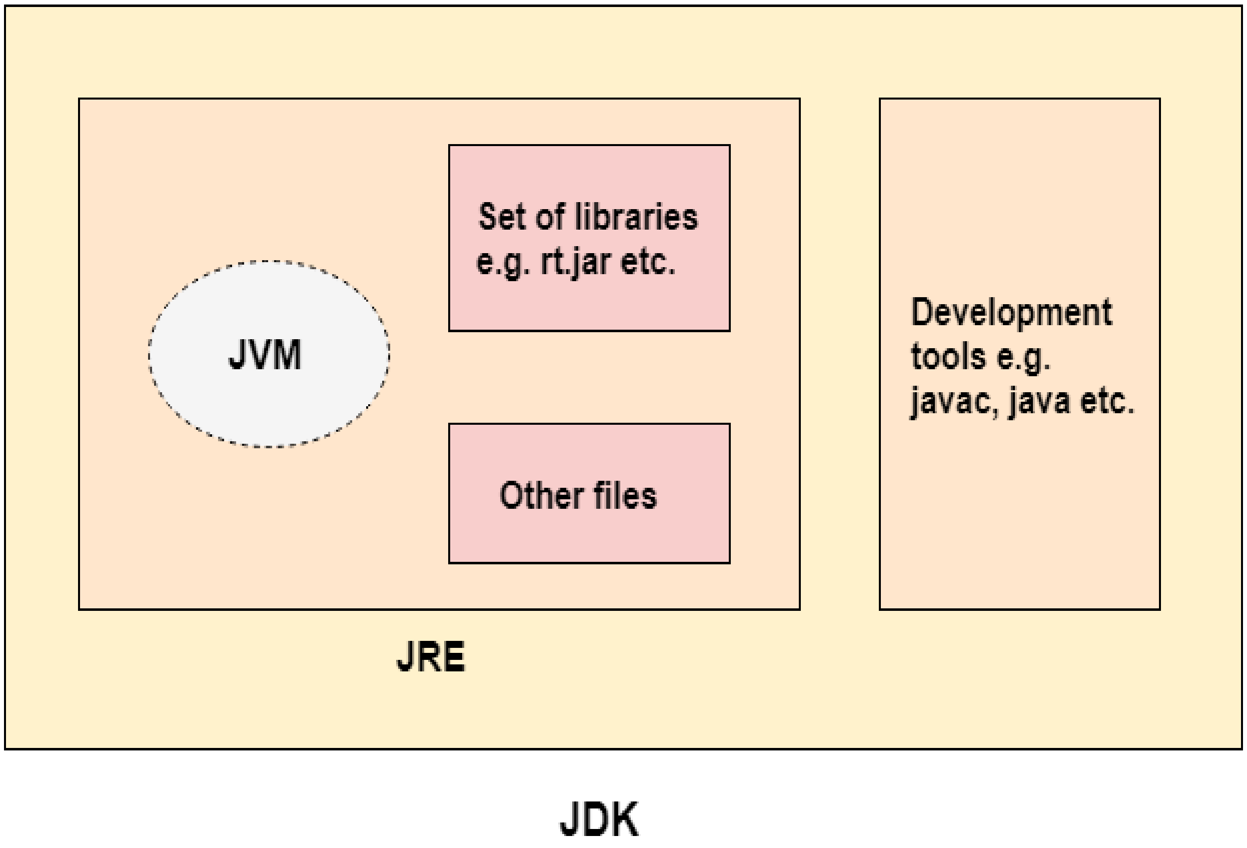
Robust:

- The Java Programming language is robust, which means it is capable of handling unexpected termination of a program.
- There are 2 reasons behind this, first, it has a most important and helpful feature called Exception Handling. If an exception occurs in java code then no harm will happen whereas, in other low-level languages, the program will crash.
- Another reason why Java is strong lies in its memory management features. Unlike other low-level languages, Java provides a runtime Garbage collector offered by JVM, which collects all the unused variables.

Different Editions in Java:

- Java Standard Edition(JavaSE)
- Java Enterprise Edition(JavaEE)
- Java Micro Edition(Java ME)

What is JDK, JRE and JVM?


Set of libraries e.g. rt.jar etc.
JVM
Other files
Development tools e.g. javac, java etc.
JRE
JDK

JDK:

- JDK is an acronym for Java Development Kit.
- The Java Development Kit (JDK) is a software development environment which is used to develop Java applications.
- It physically exists. It contains JRE + development tools.

JRE:

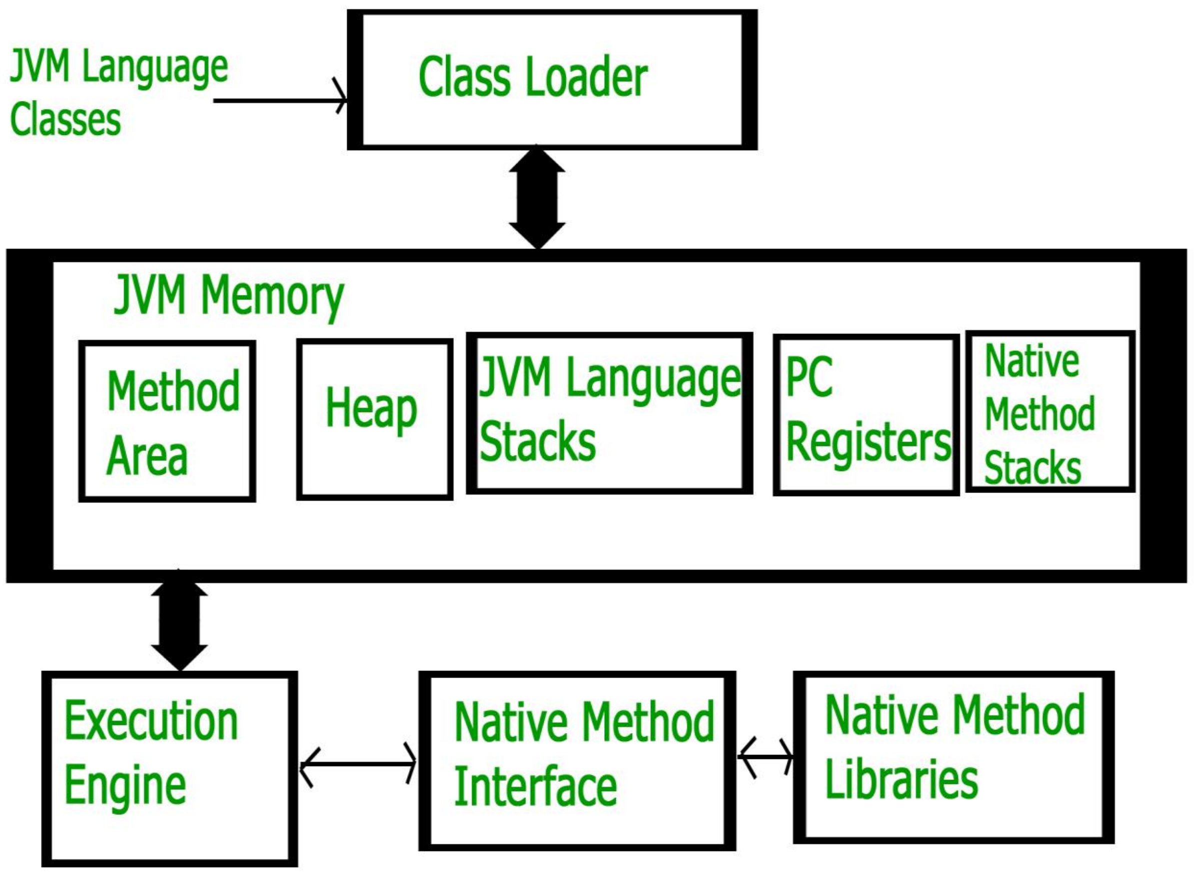
- JRE is an acronym for Java Runtime Environment. It is also written as Java RTE.
- It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JVM:

- JVM (Java Virtual Machine) is an abstract machine.
- It is called a virtual machine because it doesn't physically exist.
- It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JVM Architecture

- JVM(Java Virtual Machine) acts as a run-time engine to run Java applications.
- JVM is the one that actually calls the **main** method present in a java code.
- JVM is a part of JRE(Java Runtime Environment).
- When we compile a *.java* file, *.class* files(contains byte-code) with the same class names present in *.java* file are generated by the Java compiler.
- This *.class* file goes into various steps when we run it.



Class Loader Subsystem:

It is mainly responsible for three activities.

- Loading
- Linking
- Initialization

Loading:

- The Class loader reads the "*.class*" file, generate the corresponding binary data and save it in the method area.

- For each "*.class*" file, JVM stores the following information in the method area.

  - The fully qualified name of the loaded class and its immediate parent class.
  - Whether the "*.class*" file is related to Class or Interface or Enum.
  - Modifier, Variables and Method information etc.

Linking:

Performs verification, preparation, and (optionally) resolution.

- **Verification:**
  - It ensures the correctness of the .class file i.e. it checks whether this file is properly formatted and generated by a valid compiler or not.
  - If verification fails, we get run-time exception java.lang.VerifyError. This activity is done by the component ByteCodeVerifier. Once this activity is completed then the class file is ready for compilation.

- **Preparation:**

  JVM allocates memory for class static variables and initializing the memory to default values.

- **Resolution:**

  It is the process of replacing symbolic references from the type with direct references. It is done by searching into the method area to locate the referenced entity.

Initialization:

- In this phase, all static variables are assigned with their values defined in the code and static block(if any).

- This is executed from top to bottom in a class and from parent to child in the class hierarchy.

JVM Memory

- **Method area:** In the method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables. There is only one method area per JVM, and it is a shared resource. From java 8, static variables are now stored in Heap area.
- **Heap area:** Information of all objects is stored in the heap area. There is also one Heap Area per JVM. It is also a shared resource.
- **Stack area:** For every thread, JVM creates one run-time stack which is stored here. Every block of this stack is called activation record/stack frame which stores methods calls. All local variables of that method are stored in their corresponding frame. After a thread terminates, its run-time stack will be destroyed by JVM. It is not a shared resource.
- **PC Registers:** Store address of current execution instruction of a thread. Obviously, each thread has separate PC Registers.
- **Native method stacks:** For every thread, a separate native stack is created. It stores native method information.

Execution Engine

- Execution engine executes the “*.class*” (bytecode).
- It reads the byte-code line by line, uses data and information present in various memory area and executes instructions.
- It can be classified into three parts:
    - Interpreter: It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is required.

- Just-In-Time Compiler(JIT) : It is used to increase the efficiency of an interpreter. It compiles the entire bytecode and changes it to native code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not required, thus efficiency is improved.

- Garbage Collector: It destroys un-referenced objects.

Java Native Interface (JNI):

- ➢ It is an interface that interacts with the Native Method Libraries and provides the native libraries(C, C++) required for the execution.
- ➢ It enables JVM to call C/C++ libraries and to be called by C/C++ libraries which may be specific to hardware.

Native Method Libraries :

- ➢ It is a collection of the Native Libraries(C, C++) which are required by the Execution Engine.