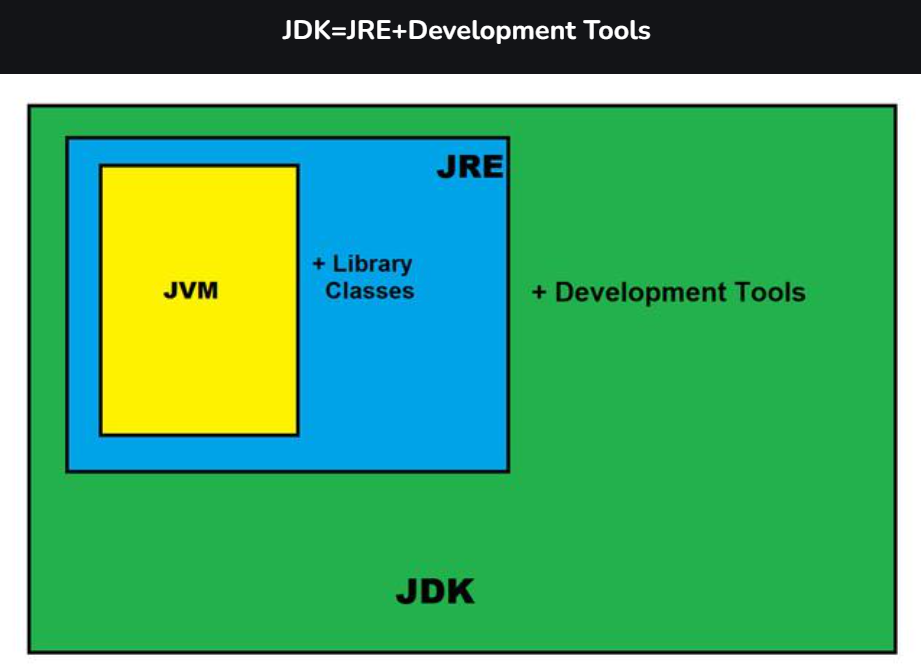
In programming languages like C and C++, the code is first compiled into platform-specific machine code. These languages are called *compiled languages*.

On the other hand, in languages like JavaScript and Python, the computer executes the instructions directly without having to compile them. These languages are called *interpreted languages*.

Java uses a combination of both techniques. Java code is first compiled into byte code to generate a *class* file. This *class* file is then interpreted by the Java Virtual Machine for the underlying platform. The same *class* file can be executed on any version of JVM running on any platform and operating system.

*Similar to virtual machines,* **the JVM creates an isolated space on a host machine. This space can be used to execute Java programs irrespective of the platform or operating system of the machine.**



### **JRE**

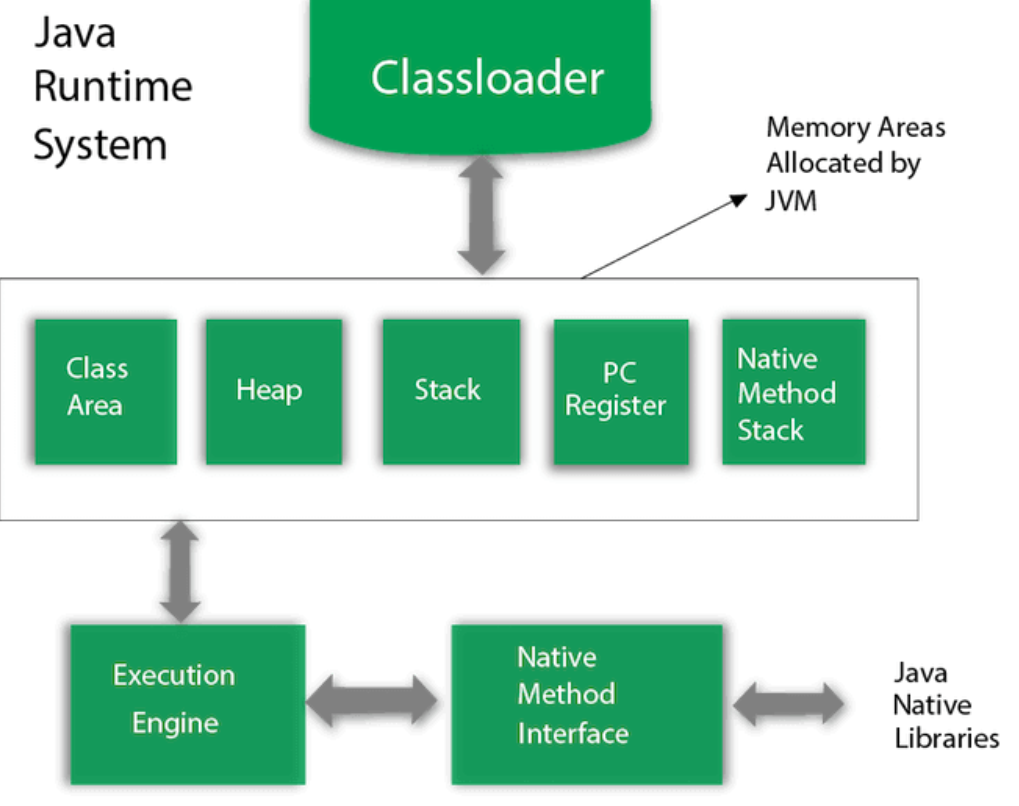
 It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

he Java Development Kit (JDK) is a software development environment which is used to develop Java applications and [applets](https://www.javatpoint.com/java-applet). It physically exists. It contains JRE + development tools.

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.

**JDK contains:**

* Java Runtime Environment (JRE),
* An interpreter/loader (Java),
* A compiler (javac),
* An archiver (jar) and many more.

****

### **1) Classloader**

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

### **2) Class(Method) Area**

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

### **3) Heap**

It is the runtime data area in which objects are allocated.

### **4) Stack**

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

### **5) Program Counter Register**

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

### **6) Native Method Stack**

It contains all the native methods used in the application.

### **7) Execution Engine**

It contains:

1. **A virtual processor**
2. **Interpreter:** Read bytecode stream then execute the instructions.
3. **Just-In-Time(JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

### **8) Java Native Interface**

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

**Java Syntax**

public class Main {

public static void main(String[] args) {

System.out.println("Hello World");

}

}

How to run with cmd =>

**C:\Users\Your Name>javac Main.java**

**C:\Users\Your Name>java Main**

**In Java, every application begins with a class name, and that class must match the filename.**

**A class should always start with an uppercase first letter.**

**Java is case-sensitive: "MyClass" and "myclass" has different meaning.**

There is also a print() method, which is similar to println().

The only difference is that it does not insert a new line at the end of the output:

## Java Variables

Variables are containers for storing data values.

In Java, there are 5 different **types** of variables, for example:

* String
* int -
* float - stores floating point numbers, with decimals, such as 19.99 or -19.99 **float mynum = 5.99f; double mynum =19.99d;**
* char -
* boolean -

The general rules for naming variables are:

* Names can contain letters, digits, underscores, and dollar signs
* Names must begin with a letter
* Names should start with a lowercase letter, and cannot contain whitespace
* Names can also begin with $ and \_ (but we will not use it in this tutorial)
* Names are case-sensitive ("myVar" and "myvar" are different variables)
* Reserved words (like Java keywords, such as int or boolean) cannot be used as names

Data types are divided into two groups:

* Primitive data types - includes byte, short, int, long, float, double, boolean and char
* Non-primitive data types - such as [String](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp) and [Classes](https://www.w3schools.com/java/java_classes.asp) (you will learn more about these in a later chapter)

Non-primitive data types are called **reference types** because they refer to objects.

A primitive data type specifies the size and type of variable values, and it has no additional methods.

* A primitive type has always a value, while non-primitive types can be null.
* A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.

## Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

* **Widening Casting** (automatically) - converting a smaller type to a larger type size  
  byte -> short -> char -> int -> long -> float -> double
* **Narrowing Casting** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char -> short -> byte
* Math.max(x,y)
* Math.min(x,y)
* Math.abs(x)
* Math.random()

**The Math.abs(x) method returns the absolute (positive) value**

**Math.random() returns a random number between 0.0 (inclusive), and 1.0 (exclusive)**

**ternary operator**

variable *= (*condition*) ?* expressionTrue *:*  expressionFalse*;*

## For-Each Loop

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

System.out.println(i);

}

## Java User Input

The Scanner class is used to get user input, and it is found in the java.util package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings:

import java.util.Scanner; // Import the Scanner class

import java.util.\*;

class Main{

public static void main(String[] args){

System.out.println("Hi");

Scanner sc = new Scanner(System.in);

String name;

System.out.println("Enter your Name");

name = sc.nextLine();

System.out.println("Your Name is: " + name);

}

}

nextInt() = for taking next int as input.

String to Int.

   String s="200";

**int** i=Integer.parseInt(s);