# Object Oriented Programming using Java

## What is Java?

Java is a programming language and a computing platform for application development. It was first released by Sun Microsystems in 1995 and later acquired by Oracle Corporation. It is one of the most used programming languages.

## What is Java Platform?

Java platform is a collection of programs that help to develop and run programs written in the Java programming language. Java platform includes an execution engine, a compiler, and a set of libraries. JAVA is platform-independent language. It is not specific to any processor or operating system.

## Brief History of Java

Java language project initially started in June 1991 by James Gosling, Mike Sheridan, and Patrick Naughton. An oak tree stood outside Gosling's office at that time and java named as oak initially. It later renamed as Green and was later renamed as java from java coffee.

## Base concept of java language project

Write once, run anywhere (WORA) – that means java program can run anywhere and on any platform. When java code is compiled it is converted into byte code. Now only this byte code is needed to run using JVM, no need of source code and recompilation.

Java released versions:

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)
11. Java SE 9 (21th Sept, 2017)
12. Java SE 10 (20th March, 2018)

## How is Java Platform Independent?

Like C compiler, Java compiler does not produce native executable code for a particular machine. Instead, Java produces a unique format called byte code. It executes according to the rules laid out in the virtual machine specification.
Byte code is understandable to any JVM installed on any OS. In short, the java source code can run on all operating systems.

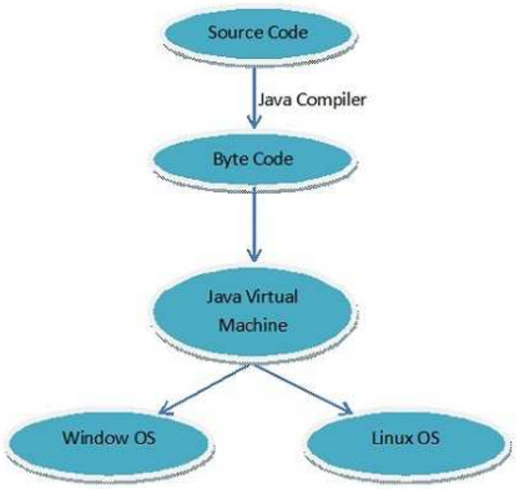
## Why java is used - Java features -

Java is used because of following features.

**1. Simple, easy and familiar:**

Java is easy to learn and familiar because java syntax is just like c++.
It is simple because:
a) It does not use header files.
b) Eliminated the use of pointer and operator overloading.

**2. Platform Independent:**
Write once, run anywhere (WORA).



**3. Object-Oriented:**

Java is Object oriented throughout language- that mean no coding outside of class definitions, including main ().

**4. Robust:**

Robust means inbuilt capabilities to handle errors/exceptions.

*Java is robust because of following:*

1. Built-in Exception handling.
2. Strong type checking i.e. all data must be declared an explicit type.
3. Local variables must be initialized.
4. Automatic garbage collection.
5. First checks the reliability of the code before Execution etc.

**5. Secure:**

Java is secure because it provides:
1. Access restrictions with the help of access modifiers (public, private etc).
2. Byte codes verification – checks classes after loading.

**Class loader** – Confines objects to unique namespaces.

**Security manager** – Determines what resources a class can access such as reading and writing to the local disk.

**6. Distributed:**

Java provides the network facility. i.e. programs can be access remotely from any machine on the network rather than writing program on the local machine. HTTP and FTP protocols are developed in java.

**7. Compiled and interpreted:**

Java code is translated into byte code after compilation and the byte code is interpreted by JVM (Java Virtual Machine). This two steps process allows for extensive code checking and also increase security.

**8. Portable:**

Means able to be easily carried or moved. Write once, run anywhere (WORA) feature makes it portable.

**9. Architecture-Neutral:**

Java code is translated into byte code after compilation which is independent of any computer architecture; it needs only JVM (Java Virtual Machine) to execute.

**10. High performance:**

JVM can execute byte codes (highly optimized) very fast with the help of Just in time (JIT) compilation technique.

**11. Re-usability of code:**

Java provides the code reusability With the Help of Inheritance.

**12. Multithreading:**

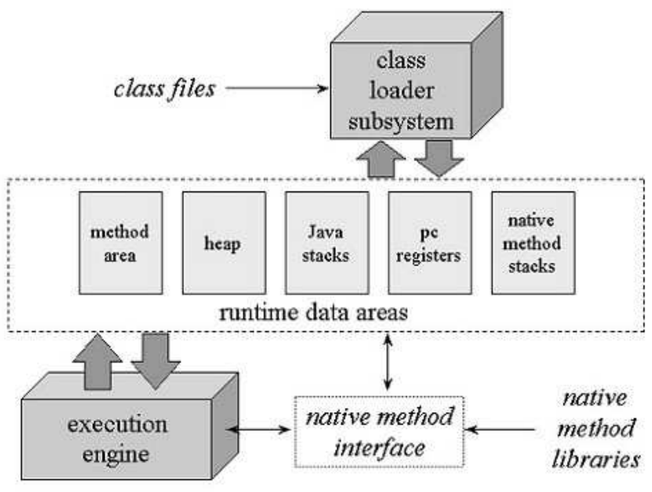
Java provides multitasking facility with the help of lightweight processes called threads.

**13. Dynamic:**

Java has the capability of linking dynamic new classes, methods and objects.

- **What is JVM?**

JVM is a virtual machine or a program that provides run-time environment in which java byte code can be executed. JVMs are available for many hardware and software platforms. The use of the same byte code for all JVMs on all platforms make java platform independent.



JVM Diagram

**JVM details:**

**1. Class loader subsystem:**

It is a part of JVM that care of finding and loading of class files.

**2. Class/method area:**

It is a part of JVM that contains the information of types/classes loaded by class loader. It also contains the static variable, method body etc.

**3. Heap:**

It is a part of JVM that contains object. When a new object is created, memory is allocated to the object from the heap and object is no longer referenced memory is reclaimed by garbage collector.

**4. Java Stack:**

It is a part of JVM that contains local variable, operands and frames. To perform an operation, Byte code instructions takes operands from the stack, operate and then return the result in to the java stack.

**5. Program Counter:**

For each thread JVM instance provide a separate program counter (PC) or pc register which contains the address of currently executed instruction.

**6. Native method stack:**

As java program can call native methods (A method written in other language like c). Native method stack contains these native methods.

**7. Execution Engine:**

It is a part JVM that uses Virtual processor (for execution), interpreter (for reading instructions) and JIT (Just in time) compiler (for performance improvement) to execute the instructions.

- **How JVM is created (Why JVM is virtual):**

When JRE installed on your machine, you got all required code to create JVM. JVM is created when you run a java program.

**Lifetime of JVM:**

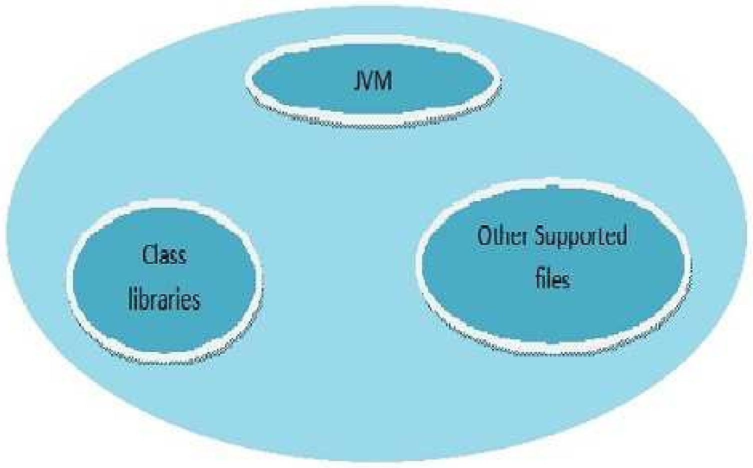
When an application starts, a runtime instance is created. When application ends, runtime environment destroyed. If n no. of applications starts on one machine then n no. of runtime instances are created and every application run on its own JVM instance.

**Main task of JVM:**

1. Search and locate the required files.
2. Convert byte code into executable code.
3. Allocate the memory into ram
4. Execute the code.
5. Delete the executable code.

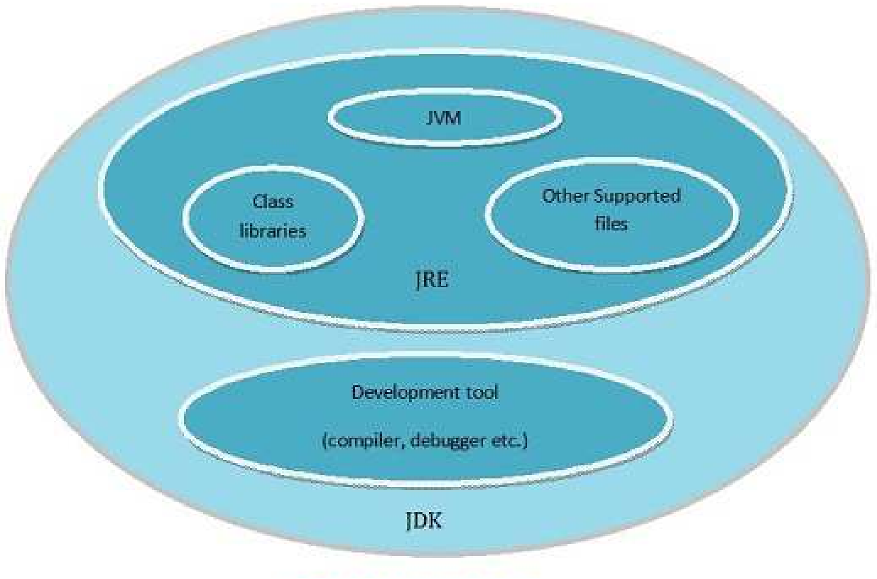
## ➢ JRE (Java RunTime Environment):

JVM + java runtime libraries + java package classes (e.g. util, lang etc). JRE provides class libraries and other supporting files with JVM. It not provide any development tool like compiler, debugger etc.



## ➢ JDK (Java Development Kit):

JRE + development tool (compiler, debugger etc.). JDK contains tools to develop the application and JRE to execute the application.



- **What is OOPS?**

Object Oriented Programming is a programming concept that works on the principle that objects are the most important part of your program. It allows users create the objects that they want and then create methods to handle those objects. Manipulating these objects to get results is the goal of Object Oriented Programming.

Object Oriented Programming popularly known as OOP, is used in a modern programming language like Java.

- **Core OOPS concepts are --**

1) **Class =** The class is a group of similar entities. It is only a logical component and not the physical entity. For example, if you had a class called "Expensive Cars" it could have objects like Mercedes, BMW, Toyota, etc. Its properties (data) can be price or speed of these cars. While the methods may be performed with these cars are driving, reverse, braking etc.

2) **Object =** An object can be defined as an instance of a class, and there can be multiple instances of a class in a program. An Object contains both the data and the function, which operates on the data. For example - chair, bike, marker, pen, table, car, etc.

3) **Inheritance =** Inheritance is an OOPS concept in which one object acquires the properties and behaviors of the parent object. It's creating a parent-child relationship between two classes. It offers robust and natural mechanism for organizing and structure of any software.

4) **Polymorphism =** Polymorphism refers to the ability of a variable, object or function to take on multiple forms. For example, in English, the verb "run" has a different meaning if you use it with "a laptop," "a foot race", and "business" also. Here, we understand the meaning of "run" based on the other words used along with it. The same also applied to Polymorphism.

5) **Abstraction =** An abstraction is an act of representing essential features without including background details. It is a technique of creating a new data type that is suited for a specific application. For example, while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about parts like steering wheel, gears, accelerator, etc.

6) **Encapsulation =** Encapsulation is an OOP technique of wrapping the data and code. In this OOPS concept, the variables of a class are always hidden from other classes. It can only be accessed using the methods of their current class. For example - in school, a student cannot exist without a class.

7) **Association =** Association is a relationship between two objects. It defines the diversity between objects. In this OOP concept, all objects have their separate lifecycle, and there is no owner. For example, many students can associate with one teacher while one student can also associate with multiple teachers.

8) **Aggregation =** In this technique, all objects have their separate lifecycle. However, there is ownership such that child object can't belong to another parent object. For example consider class/objects department and teacher. Here, a single teacher can't belong to multiple departments, but even if we delete the department, the teacher object will never be destroyed.

9) **Composition =** A composition is a specialized form of Aggregation. It is also called "death" relationship. Child objects do not have their lifecycle so when parent object deletes all children object will also delete automatically. For that, let's take an example of House and rooms. Any house can have several rooms. One room can't become part of two different houses. So, if you delete the house room will also be deleted.

## Advantages of OOPS --

- OOP offers easy to understand and a clear modular structure for programs.
- Objects created for Object-Oriented Programs can be reused in other programs. Thus it saves significant development cost.
- Large programs are difficult to write, but if the development and designing team follow OOPS concept then they can better design with minimum flaws.
- It also enhances program modularity because every object exists independently.

## What is a Variable?

A variable can be thought of as a container which holds value for you, during the life of a Java program. Every variable is assigned a data type which designates the type and quantity of value it can hold.

In order to use a variable in a program we to need to perform 2 steps --

- Variable Declaration
- Variable Initialization

## Types of Variables

In Java, there are three types of variables:

- Local Variables
- Instance Variables
- Static Variables

**Local Variables**

Local Variables are a variable that are declared inside the body of a method.

**Instance Variables**

Instance variables are defined without the STATIC keyword .They are defined Outside a method declaration. They are Object specific and are known as instance variables.

**Static Variables**

Static variables are initialized only once, at the start of the program execution. These variables should be initialized first, before the initialization of any instance variables.

e.g.,

```
class Demo
{
        int data = 99;                 //instance variable
        static int a = 1;              //static variable

        void method()
        {
            int b = 90;                //local variable
        }
}
```

➢ **Data Types in Java**

Data types classify the different values to be stored in the variable. In java, there are two types of data types:

1. Primitive Data Types
2. Non-primitive Data Types

➢ **Primitive Data Types**

Primitive Data Types are predefined and available within the Java language. Primitive values do not share state with other primitive values.

There are 8 primitive types:

byte, short, int, long, char, float, double, and boolean

**Integer data types**

| | | |
|---|---|---|
| o byte | - | 1 byte |
| o short | - | 2 bytes |
| o int | - | 4 bytes |
| o long | - | 8 bytes |

**Float data types**

| | | |
|---|---|---|
| o float | - | 4 bytes |
| o double | - | 8 bytes |

**Textual data types**

| | | |
|---|---|---|
| o char | - | 2 bytes |

**Logical data types**

| | | | |
|---|---|---|---|
| o boolean | - | 1 byte | (true/false) |

**Java Data Types**

| **Data Type** | **Default Value** | **Default size** |
|---|---|---|
| byte | 0 | 1 byte |
| short | 0 | 2 bytes |
| int | 0 | 4 bytes |
| long | 0L | 8 bytes |
| float | 0.0f | 4 bytes |

| | | |
|---|---|---|
| double | 0.0d | 8 bytes |
| boolean | false | 1 bit |
| char | '\u0000' | 2 bytes |

**Points to Remember:**

- All numeric data types are signed(+/-).
- The size of data types remain the same on all platforms (standardized)
- char data type in Java is 2 bytes because it uses **UNICODE** character set. By virtue of it, Java supports internationalization. UNICODE is a character set which covers all known scripts and language in the world.

➢ **Java Variable Type Conversion & Type Casting**

A variable of one type can receive the value of another type. Here there are 2 cases -

**Case 1:** Variable of smaller capacity is be assigned to another variable of bigger capacity.

```
double d;
int i = 10;
d = i;
```

This process is Automatic, and non-explicit is known as Conversion.

**Case 2:** Variable of larger capacity is be assigned to another variable of smaller capacity**.**

```
double d = 10;
int i;
i = (int)d;
```

In such cases, you have to explicitly specify the **type cast operator**. This process is known as **Type Casting**.

➢ **Example:** To Understand Type Casting

```
class Demo
{
        public static void main(String args[])
        {
                byte x;
                int a = 270;
                double b = 128.128;
                System.out.println ("int converted to byte");
                x = (byte) a;
                System.out.println ("a and x " + a + " " + x);
                System.out.println ("double converted to int");
                a = (int) b;
                System.out.println ("b and a " + b + " " + a);
                System.out.println ("\ndouble converted to byte");
                x = (byte) b;
                System.out.println ("b and x” + b + " "+ x);
        }
}
```

**Note:**

- Java Class is an entity that determines how an object will behave and what the object will contain.

- A Java object is a self-contained component which consists of methods and properties to make certain type of data useful.

- A class system allows the program to define a new class (derived class) in terms of an existing class (super class) by using a technique like inheritance, overriding and augmenting.