Java

Contents

James Arthur Gosling, a computer scientist from Canada, created Java Programming language in 1995. James Arthur Gosling born on born May 19, 1955 and is referred as Dr Java.
Gosling worked in Sun Microsystems for 26 long years starting from 1984 to 2010.
Naturally Sun Microsystems was owner of Java.
Oracle has owned Sun Microsystems on 27th January 2010.
Now Oracle is the owner of Java.

There are several benefits of using Java language

- It is open source and free
- It is easy to learn and implement
- It is secure,robust and powerful
- It is an object oriented and high level language
- It is a platform independent and portable.
  Java programs can be developed on one OS and can run on any OS.
  It is known as WORA (Write Once and Run Anywhere).
  It can run on various operating systems like Windows, Linux, Mac, Rasperry Pi etc.
- Through Java programming language one can make desktop applications, Web applications, and Mobile applications.

Environment and Tools needed to do Java Programming

a) It is always preferable to use a high speed machine with a configuration of processor speed 3.6 Gz and RAM of 8GB or even 16GB is good.
   One should not think that above configuration is must but as a professional it is always recommendable to use high speed machine.
b) Install latest version of JDK (Java Development Kit)
c) Latest version of IDE (Integrated Development Environment) tools like IntelliJ, Eclipse, or NetBeans.

After installing latest version java in your machine, you need to do following set up

a) Right-click on 'My Computer' and select 'Properties'.
b) Under the 'Advanced' tab, click 'Environment variables' button
c) Alter the 'Path' variable so that it also contains the path to the Java executable.
   Let us assume that Java is installed in D:\Java\Jdk\bin.
   In your local machine if the path is set to 'C:\WINDOWS\SYSTEM32',
   Then you need to add path for java.After adding the path of java executable
   It should like 'C:\WINDOWS\SYSTEM32; D:\Java\Jdk\bin'.

JVM (Java Virtual Machine)

We have already stated that Java is a high level and platform independent language.
Since it is a high level language it cannot run on the machine directly.
It needs to be translated to that machine language directly which is done by Java compiler.
The javac compiler takes java source code program which is written in .java file and translates it into machine code which is known as byte code or .class file.
JVM (Java Virtual Machine) resides in our computer and it executes the byte code and produces output. All the operating System whether it is Windows or Linux has different JVM.
Each operating system has different JVM, but they will produce same output if each of them takes same byte code. That's why Java is called platform independent language.

JRE (Java Runtime Environment)

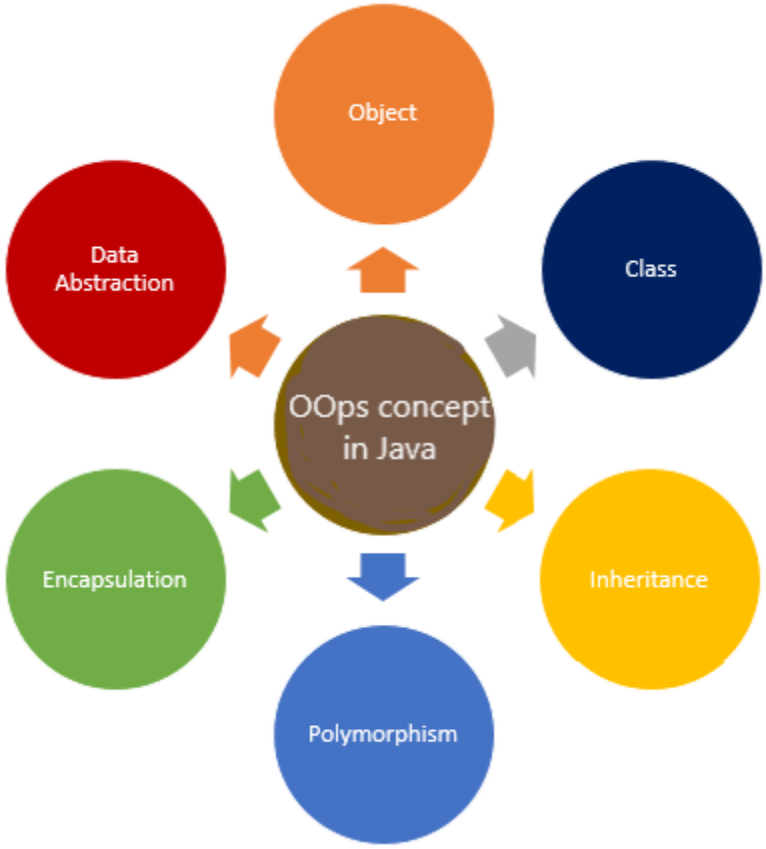
JRE stands for Java Runtime Environment. It is the implementation of JVM. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to

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.

JDK (Java Development Kit)
JDK is an acronym for Java Development Kit. It is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools. JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

Java is an object oriented programming; hence we must understand the Object Oriented Programming concept.

Object Oriented Programming system (OOPS) is defined as "it is an approach that provides a way of modularizing programs by creating partitioned memory area of both data and function (or methods) that can be used as templates for creating copies of such modules as demands".
Object-oriented programming is **a model that provides different types of concepts, such as inheritance, abstraction, polymorphism, etc.**



Objects → Objects are always called instances of a class which are created from class in java or any other language. They have states and behaviour.

Class: A class is defined as a group of objects that has similar properties (attributes), common behaviour (operations), and common relationships to other objects.

A class declaration consists of:

1. **Modifiers**: Can be private, public and protected access.
2. **Class name:** Initial letter.
3. **Superclass:** A class can only extend (subclass) one parent.
4. **Interfaces:** A class can implement more than one interface.
5. **Body:** Body surrounded by braces, { }.

Class contains properties and methods.
All the properties and methods have modifiers associated with it.

```
class Employee
{
		private Integer empId;
		private String empCode;
		private String empName;
		private String address;

		private Employee (){}
		private Employee (Integer empId, String empCode, String empName,String address)
		{
			this.empId=empId;
			this.empCode=empCode;
			this.empName=empName;
			this.address = address;
		}
		private String displayCompanyName ()
		{
			return "TCS Limited";
		}
		private void displayEmployee(Employee emp)
		{
			System.out.println("Employee Id: " + emp.empId);
			System.out.println("Employee Code: " + emp.empCode);
			System.out.println("Employee Name: " + emp.empName);
			System.out.println("Employee Address: " + emp. address);
		}
		public static void main ( String args [ ] )
		{
			Employee objEmployee=new Employee();
			System.out.println("Company Name: " +
			objEmployee.displayCompanyName());

			//Parameterized constructor
			Employee objEmp2=new Employee(1,"C01","Animesh Roy",
				"F/1 Fidder Road, Kolkata - 700012");
			objEmp2.displayEmployee(objEmp2);
		}
}
```

Java Variables
Variables stores values in Java programs.
In Java, there are different **types** of variables, for example:
String – stores text values and surrounded by double quotes, such as “India”.
int - stores integers (whole numbers), without decimals, such as 870 or -543
float - stores floating point numbers, with decimals, such as 23.42 or -15.80
char - stores single characters, such as 'q' or 'L'. Char values are surrounded by single quotes
boolean - stores values with two states: true or false

Loops in Java with example

For Loop

For loop is used to execute a set of statements repeatedly until a particular condition returns false.

```
for (int i = 1; i <= 5; i++) {
  System.out.println(i);
}
```

While Loop

In **while loop**, **condition is evaluated first** and if it returns true then the statements inside while loop execute. When condition returns false, the control comes out of loop and jumps to the next statement after while loop.

```
int i=5;
while (i>0) {
   System.out.println(i);
   i--;
}
```

This block of code, generally contains increment or decrement statements to modify the variable used in the condition. This is why after each iteration of while loop, condition is checked again. If the condition returns true, the block of code executes again else the loop ends. This way we can end the execution of while loop otherwise the loop would execute indefinitely.

Do While Loop

In while loop, condition is evaluated before the execution of loop's body but in do-while loop condition is evaluated after the execution of loop's body.

```
int i = 1;
do {
  System.out.println (i);
  i++;
}
while (i <=5);
```

Constructor

A constructor is a method used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.
Constructor name is same as that of class name.

Constructors can also take parameters, which are used to initialize attributes.
In above program both types of constructors are there.
Constructors can have parameters and may not have parameters.

Access Modifiers

In Java, access modifiers are **used to set the accessibility (visibility) of classes, interfaces, variables, methods, constructors, data members, and the setter methods**.

There are 4 access modifiers

Default	declarations are visible only within the package (package private)
Private	declarations are visible within the class only
Protected	declarations are visible within the package or all subclasses
Public	declarations are visible everywhere

```
class AccessModifiers extends ProtectedVariables
{
	public static void main(String[] args)
	{
		AccessVariables objAccessVariables = new AccessVariables();
		//Accesing Private Variable of another class is not possible.
		//Hence error will come.
		//System.out.println("Accessing private variable: " +
		objAccessVariables.deptName);
		System.out.println("Accessing default variable: " +
		objAccessVariables.designationName);
		System.out.println("Accessing public variable: " + objAccessVariables.empName);
		AccessModifiers objAccessModifiers = new AccessModifiers();
		System.out.println("Accessing protected variable: " + objAccessModifiers.cityName);
	}
}

class AccessVariables
{
	private String deptName="Accounts";
	String designationName="HR Manager";
	public String empName="Manish Sinha";
}

class ProtectedVariables
{
	protected String cityName="Kolkata";
}
```

Inheritance → Inheritance is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOPs (Object Oriented programming system).

```
public class HumanResource {
  public static void main(String[] args) {
    Manager objManager = new Manager(100, "Ramesh Singh", 25, 45000);
    objManager.printManagerDetails();
  }
}

class Employee {
```

```
 String name;
 int age;

 Employee (int age, String name) {
  this.name = name;
  this.age = age;
 }
}

class Manager extends Employee {
 int manager_id;
 int manager_salary;

 Manager (int id, String name, int age, int salary) {
  super(age, name);
  manager_id = id;
  manager_salary = salary;
 }

 void printManagerDetails() {
  System.out.println("Manager ID     :  " + manager_id);
  System.out.println("Manager Name   :  " + name);
  System.out.println("Manager Age    :  " + age);
  System.out.println("Manager Salary :  " + manager_salary);
 }
}
```

Encapsulation in Java is a process of wrapping code and data together into a single unit.
We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

```
public class AccessEmployeeType {
 public static void main(String[] args) {
        EmployeeType objEmployeeType = new EmployeeType();
        objEmployeeType.setEmployeeTypeId(1);
        objEmployeeType.setEmployeeTypeName("Permanent");
        System.out.println("Employee Type Id: " + objEmployeeType.getEmployeeTypeId());
        System.out.println("Employee Type Name: " + objEmployeeType.getEmployeeTypeName());
 }
}
class EmployeeType {
   private Integer EmployeeTypeId;
   private String EmployeeTypeName;

   public EmployeeType() {}

   public EmployeeType(Integer EmployeeTypeId, String EmployeeTypeName)
   {
      this.EmployeeTypeId = EmployeeTypeId;
      this.EmployeeTypeName = EmployeeTypeName;
   }

   public Integer getEmployeeTypeId() {
      return EmployeeTypeId;
```

```
    }

    public void setEmployeeTypeId(Integer EmployeeTypeId) {
        this.EmployeeTypeId = EmployeeTypeId;
    }

    public String getEmployeeTypeName() {
        return EmployeeTypeName;
    }

    public void setEmployeeTypeName(String EmployeeTypeName) {
        this.EmployeeTypeName = EmployeeTypeName;
    }

}
```

Polymorphism

The derivation of the word Polymorphism is from two different Greek words- poly and morphs. "Poly" means numerous, and "Morphs" means forms. So, polymorphism means innumerable forms.

Polymorphism in Java is the task that performs a single action in different ways.

There are two types of Polymorphism

a) Method Overloading → Methods having same name but different parameters.
b) Method Overriding → Parent class and child class have same method name but definition is different as functionality is different.

Method Overloading

```
class HelperService {

    private String formatNumber(int value) {
        return String.format("%d", value);
    }


    private String formatNumber(double value) {
        return String.format("%.3f", value);
    }

    private String formatNumber(String value) {
        return String.format("%.2f", Double.parseDouble(value));
    }

    public static void main(String[] args) {
        HelperService hs = new HelperService();
        System.out.println(hs.formatNumber(500));
        System.out.println(hs.formatNumber(89.9934));
        System.out.println(hs.formatNumber("550"));
    }
}
```

Method Overriding

```
public class AccessEmployee
{
  public static void main(String[] args)
  {
       Employee objEmployee  = new Employee (1, "Ramesh Singh", 25, 15000.00);
       System.out.println("General Employee salary: Rs " + objEmployee.CalculatePaySlip());

       HRManager objHRManager = new HRManager (2, "Suresh Rao", 35, 25000.00,2000.0);
       System.out.println("HR manager salary: Rs " + objHRManager.CalculatePaySlip());
  }
}

class Employee
{
      int id;
      String name;
      int age;
     Double basicSalary;

     Employee (int id,String name,int age, Double basicSalary)
     {
          this.id = id;
          this.name = name;
          this.age = age;
          this.basicSalary = basicSalary;
     }

    Double CalculatePaySlip()
    {
        Double pf= (12*basicSalary)/100;
        return basicSalary + pf;
    }
}

class HRManager extends Employee
{
    Double hrManager_Incentive;

    HRManager (int id, String name, int age, Double basicSalary,Double incentive)
    {
        super (id,name,age,basicSalary);
        hrManager_Incentive=incentive;
    }

   Double CalculatePaySlip()
   {
      Double pf= (12*basicSalary)/100;
      return basicSalary + pf + hrManager_Incentive;
   }
}
```

Abstraction

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.
Abstraction can be achieved with either **abstract classes** or **interfaces**
The `abstract` keyword is a non-access modifier, used for classes and methods:

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).
An abstract class can have both abstract and regular methods:

```
class SaveEmployee {
  public static void main(String[] args) {
    AccessEmployee objAccessEmployee = new AccessEmployee();
    System.out.println(objAccessEmployee.SaveEmployeeInfo());
  }
}

// Abstract class
abstract class Employee {
  public abstract String SaveEmployeeInfo();
}

// Subclass (inherit from Employee )
class AccessEmployee extends Employee {
  public String SaveEmployeeInfo() {
    // The body of SaveEmployeeInfo() is provided here
    return "Employee Information Saved Successfully";
  }
}
```

Interface

Interface is another abstraction in Java.
Some points related to interface

1) As Java cannot implement multiple inheritances hence interface is used.
2) Objects cannot be created through interface.
3) Interface methods do not have body. The body is provided by class it implements.
4) On implementation of an interface, you must override all of its methods
5) Interface methods are by default abstract and public
6) Interface attributes are by default public, static and final
7) An interface cannot contain a constructor (as it cannot be used to create objects)

```
class DisplayEmployee {
  public static void main(String[] args) {
    AccessEmployee objAccessEmployee = new AccessEmployee();
    System.out.println(objAccessEmployee.DisplayEmployeeInfo());
  }
}
```

```
// interface
interface Employee {
  // Abstract method
  public String DisplayEmployeeInfo();
}

// implements from Employee
class AccessEmployee implements Employee {
  public String DisplayEmployeeInfo() {
    // The body of DisplayEmployeeInfo() is provided here
    return "Show Employee Information";
  }
}
```

List

List in Java provides the facility to maintain the *ordered collection*. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.

The List interface is found in the java.util package and inherits the Collection interface. It is a factory of ListIterator interface. Through the ListIterator, we can iterate the list in forward and backward directions. The implementation classes of List interface are ArrayList, LinkedList, Stack and Vector. The ArrayList and LinkedList are widely used in Java programming.

```
import java.util.*;

public class EmployeeList
{
	public static void main(String args[])
	{
		//Creating a List
		List<String> listEmployee=new ArrayList<String>();
		//Adding elements in the List
		listEmployee.add("Ramesh Singh");
		listEmployee.add("Anuradha Roy");
		listEmployee.add("Mohit Verma");
		listEmployee.add("Gourav Chatterjee");
		//Iterating the List element using for-each loop
		for(String employee:listEmployee)
			System.out.println(employee);
	}
}
```

Exception

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

There are 3 types of Exceptions.
1) Checked Exception
The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception
The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error
Error is irrecoverable. Some examples of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

To handle exception try, catch and finally block.

Finally block can be used to clean up the code like closing file or database connection.
Finally block will execute.

```
public class ExceptionHandling
{
	public static void main(String args[])
	{
		try
		{
		    int data=50/0;
		}catch(ArithmeticException ex)
		{
			System.out.println("Exception Generated is: " + ex);
		}
		finally
		{
			System.out.println("Finally Block Executed");
		}
		System.out.println("After Exception Handling");
	}
}
```