Abstraction

Example of Abstraction –

- Driving of a car.
  - From outside car is a single object
  - Driving the car consists of functioning of various subsystems like steering, brakes, accelerators, engine etc
  - To outside person driving of a car appears much easy with a driver/person who is driving it.
  - The outside person or even the person who is driving the same he/she is unaware about the detailed functionality that is performed by the individual subsystems when the car is moving.

- Making a phone call
  - A person who is making a phone call is unaware about the different actions/functions that take place while he/she is making the phone call.

Class (Example of Encapsulation in Java)

Class defines the structure and behavior (which is data and code) that will be shared by a set of objects.

Objects of a given class contain data and code defined by that class. So objects are sometimes referred to as 'instance of class'. So class is logical construct and object has physical reality.

Class constitutes of **code** and **data**. **Code** and **data** are collectively known as **members**

Data defined by the class are known as member variables

Code that operates on data referred to as **methods**.

Methods or variables inside a class can be marked as **Private** or **Public**

Object and Class Example: main within class

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by **new** keyword and printing the objects value.

Here, we are creating main() method inside the class.

File: Student.java

```
class Student
{
    int id;//field or data member or instance variable
    String name;

 public static void main(String args[])
{
    Student s1=new Student();//creating an object of Student
    System.out.println(s1.id);//accessing member through reference variable
    System.out.println(s1.name);
 }
}
```

Output :

0

null

Inheritance

Inheritance in java is the mechanism where one object (which is the child object) derives or acquires all the properties and behavior of parent object.

```
class Subclass-name extends Superclass-name
{
    //methods and fields
}
```

The keyword **extends** indicates making of a new class from an existing class. Meaning of 'extends' is increasing of functionality.

Single Inheritance

```
public class ClassA
{
        public void dispA()
        {
                System.out.println("disp() method of ClassA");
        }

}

public class ClassB extends ClassA
{
        public void dispB()
        {
                System.out.println("disp() method of ClassB");
        }
        public static void main(String args[])
        {
                //Assigning ClassB object to ClassB reference
                ClassB b = new ClassB();
```

```
            //Call dispA() method of ClassA
            b.dispA();
            //Call dispB() method of ClassB
            b.dispB();

        }

    }
```

Output :

```
disp() method of ClassA
disp() method of ClassB
```

Multiple Inheritance

Multiple Inheritance is nothing but **one** class **extending more than** one class. Java does not support multiple inheritance as it throws compile errors. The scenario explained below –

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

```
class A
{
        void msg(){System.out.println("Hello");}
}

class B
{
    void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

    Public Static void main(String args[])
    {
        C obj=new C();
```

```
        obj.msg();//Now which msg() method would be invoked?
    }
}
```

Here the program will throw compilation error as object of class C will be in confusion as to which class msg() function to call.

Multilevel Inheritance

In **Multilevel Inheritance** a derived class will be **inheriting a parent class** and as well as the derived class **act as the parent class** to other class.

From the diagram it is seen that **Class B** is a derived class and inherits the property of **Class C**. Again **Class B** is the parent of **Class A**. In short **Class C** is the parent for **Class B** and **Class B** is the parent for **Class A**

```
public class ClassA
{
   public void dispA()
   {
      System.out.println("disp() method of ClassA");
   }
}
public class ClassB extends ClassA
{
   public void dispB()
   {
      System.out.println("disp() method of ClassB");
   }
}
public class ClassC extends ClassB
{
   public void dispC()
   {
      System.out.println("disp() method of ClassC");
   }
   public static void main(String args[])
   {
      //Assigning ClassC object to ClassC reference
      ClassC c = new ClassC();
      //call dispA() method of ClassA
```

```
        c.dispA();
        //call dispB() method of ClassB
        c.dispB();
        //call dispC() method of ClassC
        c.dispC();
    }
}
```

Output :

```
disp() method of ClassA
disp() method of ClassB
disp() method of ClassC
```

Hierarchical Inheritance

Here **ClassA** acts as the **parent** for sub classes **ClassB**, **ClassC** and **ClassD**.

```
public class ClassA
{
    public void dispA()
    {
        System.out.println("disp() method of ClassA");
    }
}
public class ClassB extends ClassA
{
    public void dispB()
    {
        System.out.println("disp() method of ClassB");
    }
}
public class ClassC extends ClassA
{
    public void dispC()
    {
        System.out.println("disp() method of ClassC");
    }
```

```
}
public class ClassD extends ClassA
{
   public void dispD()
   {
      System.out.println("disp() method of ClassD");
   }
}
public class HierarchicalInheritanceTest
{
   public static void main(String args[])
   {
      //Assigning ClassB object to ClassB reference
      ClassB b = new ClassB();
      //call dispB() method of ClassB
      b.dispB();
      //call dispA() method of ClassA
      b.dispA();


      //Assigning ClassC object to ClassC reference
      ClassC c = new ClassC();
      //call dispC() method of ClassC
      c.dispC();
      //call dispA() method of ClassA
      c.dispA();

      //Assigning ClassD object to ClassD reference
      ClassD d = new ClassD();
      //call dispD() method of ClassD
      d.dispD();
      //call dispA() method of ClassA
      d.dispA();
   }
}
```

Output :

```
disp() method of ClassB
disp() method of ClassA
disp() method of ClassC
```

```
disp() method of ClassA
disp() method of ClassD
disp() method of ClassA
```

POLYMORPHISM

Method Overloading

```
class Overloading
{
	public void disp()
	{
		System.out.println("Inside First disp method");
	}

	public void disp(String val)
	{
		System.out.println("Inside Second disp method, value is: "+val);
	}

	public void disp(String val1,String val2)
	{
		System.out.println("Inside Third disp method, values are :
"+val1+","+val2);
	}
}
public class MethodOverloading_Example
{
	public static void main (String args[])
	{
		Overloading oo = new Overloading();
		oo.disp(); //Calls the first disp method
		oo.disp("Java Interview"); //Calls the second disp method
		oo.disp("JavaInterview", "Point"); //Calls the third disp method
	}

}
```

The output will be

```
Inside First disp method
Inside Second disp method, value is: Java Interview
Inside Third disp method, values are : JavaInterview,Point
```

Here the disp() method will be called three times, but the question is how the different disp() are called. The answer is based on the parameter the compiler will choose which methods to be called.

Method Overriding

```
class ParentClass
{
        public void disp()
        {
                System.out.println("Parent Class is called");
        }
}
class ChildClass extends ParentClass
{
        public void disp()
        {
                System.out.println("Child Class is called");
        }
}
public class Overriding_Example
{
        public static void main(String args[])
        {
                ParentClass obj1 = new ParentClass();

                //ParentClass reference but ChildClass Object.
                ParentClass  obj2 = new ChildClass();

                // Parent Class disp () will be called.
                obj1.disp();
```

```
                // Child Class disp () will be called, as it reference to the child
class.
                obj2.disp();
        }
}
```

The Child Class disp is called because though the obj2 reference may be a ParentClass reference but the object is ChildClass object and hence the disp() of the child class is called . This is called as **Dynamic Binding or Late Binding or Runtime Polymorphism**