**Object Oriented Concepts in Java**

All object-oriented programming languages provide mechanisms that help us to implement the object-oriented model. They are encapsulation, inheritance, and polymorphism

**Encapsulation:** The mechanism of wrapping the data(variables) and codes which acts upon the data(methods) together as a single unit is called encapsulation. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.  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. it is a protective shield that prevents the data from being accessed by the code outside this shield.

In Java, the basis of encapsulation is the class. A class defines the structure and behavior (data and code) that will be shared by a set of objects. Each object of a given class contains the structure and behavior defined by the class, as if it were stamped out by a mold in the shape of the class. For this reason, objects are sometimes referred to as instances of a class. Thus, a class is a logical construct; an object has physical reality. When you create a class, you will specify the code and data that constitute that class. Collectively, these elements are called members of the class. Specifically, the data defined by the class are referred to as member variables or instance variables. The code that operates on that data is referred to as member methods or just methods

package boxdemo;

class Box

{

int length;

int breadth;

int height;

int volume()

{

return(length\*breadth\*height);

}

}

public class BoxDemo

{

public static void main(String[] args)

{

Box myBox=new Box();

myBox.length=10;

myBox.breadth=20;

myBox.height=30;

int vol=myBox.volume();

System.out.println("Volume of Box with given dimensions is "+vol);

}

}

**Output:**

Volume of Box with given dimensions is 6000

**Advantages of Encapsulation**:

* **Data Hiding:** The user will have no idea about the inner implementation of the class. It will not be visible to the user that how the class is storing values in the variables. He only knows that we are passing the values to a setter method and variables are getting initialized with that value.
* **Increased Flexibility:** the variables of the class can be made as read-only or write-only depending on our requirement. If we wish to make the variables as read-only then we have to omit the setter methods like setName(), setAge() etc. from the above program or if we wish to make the variables as write-only then we have to omit the get methods like getName(), getAge() etc. from the above program
* **Reusability:** Encapsulation also improves the re-usability and easy to change with new requirements.
* **Testing code is easy:** Encapsulated code is easy to test for unit testing.

**Inheritance:** Inheritance is the process by which one object acquires the properties (methods and fields) of another object. With the use of inheritance the information is made manageable in a hierarchical order (that is, top-down).

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

* **extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

//Program to demonstrate inheritance demo

package inheritancedemo;

class Super

{

int i,j;

void showij()

{

System.out.println("values of i and j are "+i+"\t"+j);

}

}

class Sub extends Super

{

int k;

void showk()

{

System.out.println("Value of k is "+k);

}

void Sum()

{

System.out.println("Sum of i & j & k is "+(i+j+k));

}

}

public class InheritanceDemo

{

public static void main(String[] args)

{

Super superobj=new Super();

Sub subobj=new Sub();

superobj.i=10;

superobj.j=20;

System.out.println("Super object contents");

superobj.showij();

subobj.i=7;

subobj.j=8;

subobj.k=9;

System.out.println("Sub object contents");

subobj.showij();

subobj.showk();

System.out.println("Sum of contents in sub is");

subobj.Sum();

}

}**Output:**

Super object contents

values of i and j are 10 20

Sub object contents

values of i and j are 7 8

Value of k is 9

Sum of contents in sub is

Sum of i & j & k is 24

**Polymerization:**