Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates.Other way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

The variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.

The data in a class is hidden from other classes, so it is also known as **data-hiding**.

Encapsulation can be achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

**Getter and Setter.**

**To access private members, we are using Getter and Setter methods.**

**Syntax:**

**public void setNumber(int n)**

**{**

**Number=n;**

**}**

**public int getNumber()**

**{**

**return(number);**

**}**

**Demo:**

import java.util.\*;

class Employee

{

private int eno;

public void setEno(int no)

{

eno=no;

}

public int getEno()

{

return(eno);

}

}

class First

{

public static void main(String[] args)

{

Employee e = new Employee();

e.setEno(101);

System.out.println("Number is :"+e.getEno());

}

}

Task:

Create a Employee Class with username,password and id as properties of following

username can read and write

password can write only property

id can read only properties.

**Advantages of Encapsulation**:

**Data Hiding**

**Increased Flexibility – Read Only and Write Only**

**Reusability**

**Object Casting**

Converting one type of object into another type of object.

## **Upcasting**

**Casting from a subclass to a superclass is called upcasting**. Typically, the upcasting is implicitly performed by the compiler.

Placing sub class memory into super class reference.

* A reference variable can refer to an object if the object is of the same type as a variable or if it is a subtype
* Upcasting happens implicitly
* All Java objects are polymorphic and can be treated as objects of supertype due to upcasting

## ****Downcasting****

**It’s the casting from a superclass to a subclass.**

**Placing sub class memory into sub class reference.**

**Demo**

**class A**

**{**

**public void m1(){}**

**}**

**class B**

**{**

**public void m1(){}**

**}**

class B

{

**public void m1(){}**

}

class CastingDemo

{

public static void main()

{

C c = new C();

B b = (B) c; // Upcasting // m1(),m2()

C cc = (C) b; // Downcasting //m1(),m2(),m3()

}

}

**VarArgs in Java**

Variable Arguments (**Varargs) in Java**. ... A method that takes a variable number of arguments is a **varargs.**

**Always last priority**

**Only one varargs**

**It is should be last parameter of method**

**Test1:**

**M1(int a,int… numbers)**

**M1(int a,int b, int… numbers)**

**M1(10,12,13) => Which one will call?**

**Test2:**

**M1(int[] a)**

**M1(int… numbers)**

**M1(10,11,12) => Which one will call?**

**Test3:**

**M1(object… obj)**

**M1(null,null,null);**

**M1(null,null);**

**M1(null);**

**M1(new Object[]{null});**