**Encapsulation:**

It is the Process of Wrapping the data members (global variable) and member function (methods) into a single unit (class) is called as “Encapsulation”

(OR)

Protecting the data members by keeping them as **Private** & accessing them through some special methods is called as “Encapsulation”

* The main purpose of Encapsulation is to achieve the Security of data
* We go for Encapsulation to protect our data members from invalid user
* If data member is protected(private), it is under developers control to decide which values to be allowed

**Rules for Encapsulation:**

1. Declare all the data members as private (if a data member is private, it is not accessible outside of class)
2. define separate **setter** and **getter** methods (it's not mandatory to define only setter and getter, we can define any user define methods)

**setter method:**

1. it is public in nature
2. setter method **does not** have specific **return type**
3. setter method contains arguments **same** as data member
4. name of method is **set** **followed by data member name**

**getter method:**

1. it is public in nature
2. return type must be **same** as that of data member
3. name of method is **get followed by data member name**
4. getter method **does not have arguments**

**Note:**

* For every data member we have to define **separate** setter and getter method

**Advantages Of Encapsulated class:**

* Securing the data
* We can make our class as **write only** by **writing only setter method**
* We can make our class as **read only** by **defining only getter method**

**Dis-advantages Of Encapsulated class:**

* length of the code increases

-------------------------------------- \*\*\*\*\* ------------------------------------------------------------------------

**Abstraction:**

It is the second pillar of OOPS concept

The process of hiding the internal implementation and showing the necessary data to the end user is called as “Abstraction”

(OR)

The process of defining a method signature & hiding the method implementation is called as “Abstraction”

In JAVA abstraction can be achieved in two ways

1. Using Abstract class
2. Using Interface

**Concrete Class:**

A Class in which all the methods are concrete such class is called as “Concrete Class”

**Ex:**

class Food

{

Public void nonveg ()

{

//body or implementation

}

}

**Abstract Class:**

* abstract is a keyword, which indicates **incompleteness**
* abstract class is a class which contains **at least one abstract method**

**Ex:**

**abstract** class Vehicle

{

**abstract** public void noOfWheels();

}

**Normal/Concrete/Complete method:**

If a method has method signature as well as method body such method is called “Complete or Concrete method”

**Ex:**

public void run ()

{

SOP("In run");

}

**Abstract/Incomplete method:**

If a method contains only method signature but not method implementation is called as abstract or incomplete method.

**Ex:**

public void run ()

public void fly ()

• Abstract method has to be represented with **abstract keyword** and we have to add **;** **(Semicolon)** in the end of method declaration.

• Abstract method **cannot be Static/final/Private.**

**Ex:**

abstract public void run ();

abstract public void fly ();

• If we didn't add abstract keyword or semicolon, we will get compile time error.

• It is mandatory to mention the abstract keyword for abstract methods and abstract classes

**Note:**

• Abstract class can have complete as well as incomplete methods.

• Abstract class can have static, non-static and final variables

**Conclusion:**

Since, Abstract class contains complete as well as incomplete methods. We can able to achieve partial abstraction i.e., 0 to 100% abstraction. Therefore, to achieve 100% abstraction, we go for **interface**

-------------------------------------- \*\*\*\*\* ------------------------------------------------------------------------

**Program For Factorial:**

**Ex:**

**public** **class** Factorial

{

**public** **static** **void** main(String[] args)

{

**int** i,fact=1; //fact=1 Assume

**for**(i=7;i>=1;i--)

{

fact=fact\*i;

}

System.***out***.println("7! is: " +fact);

}

}

**Output:**  **7!** is: **5040**

-------------------------------------- \*\*\*\*\* ------------------------------------------------------------------------

**Program For Fibonacci Series:**

A series of Numbers whose 1st two terms are 0 & 1 and from 3rd term **every number is sum of Previous two numbers**.

**Ex:** 0 1 1 2 3 5 8 13………………

**public** **class** FibonacciSeries

{

**public** **static** **void** main(String[] args)

{

**int** f=0,f1=1,f2;

System.***out***.print(f+" ");

System.***out***.print(f1+" ");

**for**(**int** i=1;i<=6;i++) //except 0 & 1 we are adding 6 Numbers

{

f2=f+f1;

System.***out***.print(f2+" ");

f=f1;

f1=f2;

}

}

}

**Output:** **0 1 1 2 3 5 8 13**

-------------------------------------- \*\*\*\*\* ------------------------------------------------------------------------