24.01.24

# JAVA OOPS

Q What is OOP ?

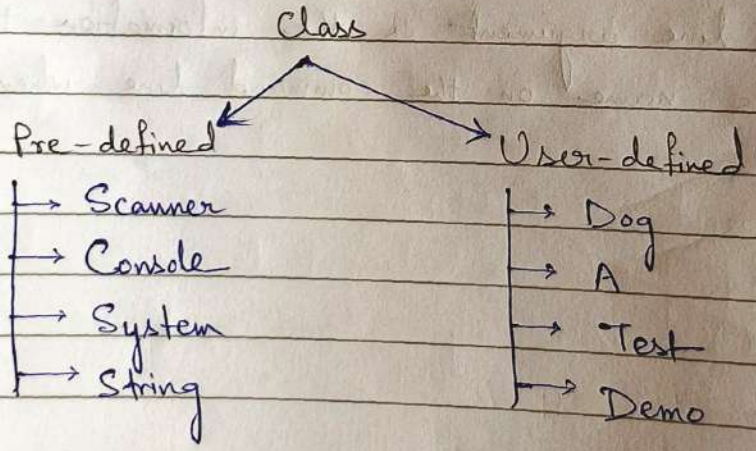
Ans OOP stands for Object Oriented Programming language, the main purpose of OOP is to deal with real world entity using programming language.

OOPS Features :

- Class
- Object
- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

Q What is Class ?

Ans Class is a collection of objects and it doesn't take any space on memory. Class is also called as blueprint / logical entity.




User-defined Class :

A class which is created by Java programmer is called user-defined class.

ex :
```
class class_name
{
    // data
    // methods
}
```

Q What is Object ?

Ans Object is an instance of class that executes the class. Once the object is created, it takes up space like other variable in memory.

Syntax :

```
class_name  obj_name  =  new  class_name()
```

- class_name → class name
- obj_name → object reference
- new → Dynamic Memory Allocation
- class_name() → Constructor

Sample Program

```
class Demo
{
    int a = 20 , String b = "Anindya" ;
    void show()
    {
        System.out.print(a+" "+b);
    }
}
class Test
{
    public Static void main(String[] args)
    {
        Demo r = new Demo();
        r.show();
    }
}
```

O/p : 20 Anindya

Q What is Constructor ?

Ans Constructor is a special type of method whose name is same as class name.

I) The main purpose of constructor is to initialize the object.

II) Every java class has a constructor.

III) A constructor is automatically called at the time of object creation.

IV) A constructor never contain any return type including void.

Syntax :

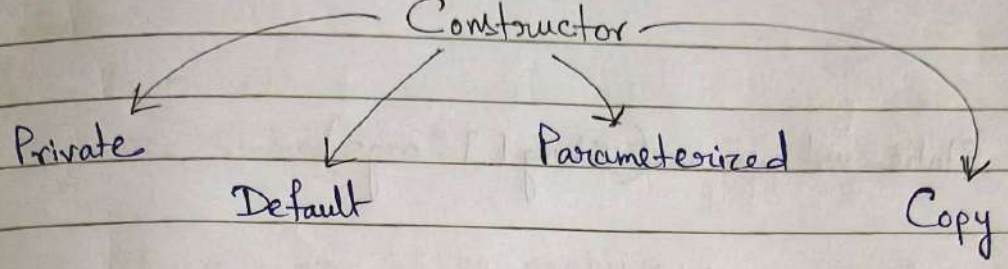
```
class class_name
{
    class_name()
    {
      - -
      - -
    }
}
```

```
class A
{ int a, String name;
  A()
  {
    a=0, name = null;
  }
  void show()
  {
    System.out.print(a+" "+name);
  }
}
class B
{
  A ref = new A();
  ref.show();
}
```

if you don't add this constructor or don't initialize the values, java compiler will automatically create and set the initialized value.

Output :

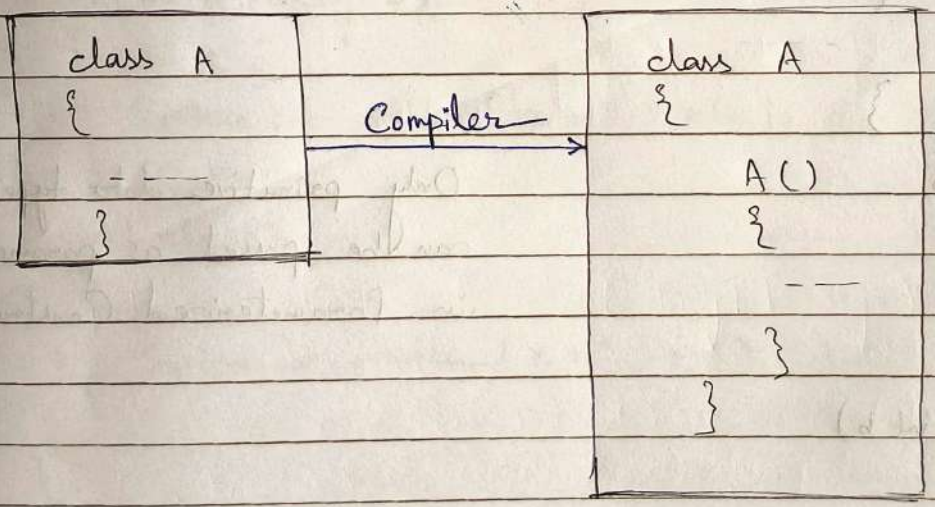
0 null




## Default Constructor :

A constructor which does not have any parameter is called Default Constructor.

Syntax :
```
class A
{
    A()        → No parameter
    {
      ---
    }
}
```




```
class A
{ int a; String name; boolean c;
  A()
  {
     a = 100 ; name = "Arindya" ; c = true ;
  }
  void show()
  {
     System.out.print(a+" "+b+" "+c);
  }
}
```

if default constructor is not defined it'll print a = 0, name = null, c = false

```
class B
{
   public Static void main( String [] args)
   {
      A ref = new A();
      ref.show();
   }
}
```

Output

100 Anindya true

Parameterized Constructor :

A constructor through which we can pass one or more parameters is called parameterized constructor.

Syntax :

```
class A
{
    A ( int X, String Y)
    {
       ----
    }
}
```

Only primitive data types can be passed as parameters in Parameterized Constructor

```
class A
{ int x,y;
   A(int a, int b)
   {
      x=a; y=b;
   }
   void show()
   {
      System.out.print(x+" "+y);
   }
}
```

```
class B
{
   public Static void main (String [] args)
   {
      A r = new A(100, 200);
      r.show();
   }
}
```

Output :
100 200

```
class A
{
   int x, y;
   A (int a, int b)
   {
      x = a ; y = b ;
   }
   A (int a, String b)
   {
      System.out.println ('a + " " + b);
   }
   void show()
   {
      System.out.println (x + " " + y);
   }
}
class B
{ public Static void main (String[] args){
   A r = new A (100, 200);
   r.show();
   A ref = new A (100, "Anindya");
   }
}
```

Output
100 200
100 Anindya

## Copy Constructor :

Whenever we pass object reference to the constructor then it is called Copy Constructor.

Syntax :
```
class class_name
{
    class_name (obj ref)
    {
        ---
    }
}
```

Copy Constructor is used to copy all the contents of one constructor into another.

```
class A
{
  int a; String b;
  A()
  {
    a=10 ; b = "LearnCoding";
    System.out.println (a+" " + b);
  }
  A(A ref)
  {
    a = ref.a ;
    b = ref.b;
    System.out.println (a+" " + b);
  }
}
class B
{
  public static void main (String[] args)
  {
    A r1 = new A();
    A r2 = new A(r1);
  }
}
```

Output

10 LearnCoding
10 LearnCoding

## Private Constructor :

In Java it is possible to write a constructor as a private but according to the rule we can't access private members outside of class.

Syntax :

```
class class_name
{
    private class_name()
    {
        ---
    }
}
```

```
class A
{
  int a; float b; String c;
  private A()
  {
    a=10; b=30.56; c="Anindya";
    System.out.println(a+" "+b+" "+c);
  }
  public static void main(String[] args)
  {
    A r = new A();
  }
}
```

Output

10 30.56 Anindya

if you create the object outside class A (in a newly created class suppose class B) it'll show an error because the constructor has private access and the content inside the constructor is only accessible inside the class A.

## Constructor Overloading:

Whenever we have more than one constructor in our class, then, it is called Constructor Overloading.

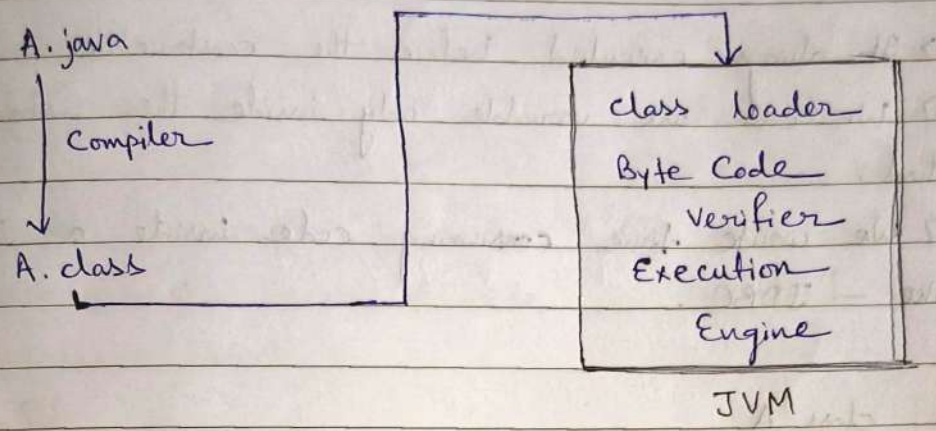
```
class A
{
   int a; double b; String c;
   A()
   {
      a=100; b=45.32; c="Anindya";
   }
   A(int x)
   {
      a=x;
   }
   A(double Y, String Z)
   {
      b=Y; c=Z;
   }
}
class B
{
   public static void main(String[] args)
   {
      A r = new A();
      A r2 = new A(10);
      A r3 = new A(23.89, "Anindya");
      System.out.println(r.a+" "+r.b+" "+r.c);
      System.out.println(r2.a);
      System.out.println(r3.b+" "+r3.c);
   }
}
```

O/P

```
100 45.32 Anindya
10
23.89 Anindya
```

## Static Block

Static block is such kind of block in Java which gets executed at the time of loading the .class file into the JVM memory.




Syntax :
```
class A
{
    static
    {
      ----
    }
}
```

```
class A
{
  public static void main (String[] args)
  {
     A r = new A();
  }
  {
    System.out.println("My name is");
  }
  A()
  {
    System.out.println("Anindya");
  }
  static {
    System.out.println("Hello everyone");
  }
```

Instance block

Normal Constructor

static block

Execution Order :

① Static
② Instance
③ Constructor

O/P

Hello everyone
My name is
Anindya

# Instance Block

Instance block is similar to method which has no name, it can be written inside a class only but not inside a method.

Note :

i) It always executed before the constructor.

ii) We can use variable only inside the instance block not method.

iii) We write time consuming code inside a instance block like - JDBC.

Syntax :
```
class A
{
    {
        ---
    }
}
```

```
class A
{
  int a,b;
  A()
  {
    a = 30; b = 40;
    System.out.println(a+" "+b);
  }
  {
    a = 10; b = 20;
    System.out.println(a+" "+b);
  }
}
class B
{
  public static void main(String[] args){
    A r = new A();
  }
}
```

o/p

10 20

30 40

# Instance Vs Static Block

| Instance | Static |
|---|---|
| (i) It deals with object. | (i) It deals with class. |
| (ii) Executed at the time of object creation. | (ii) Executed at the time of loaded .class file in JVM. |
| (iii) No keyword required. | (iii) Static keyword is required. |
| (iv) Static & non-static variable can be accessed inside the instance block. | (iv) Only static variable can be accessed inside the static block. |

## Inheritence

When we construct a new class from existing class in such a way that the new class access all the features and properties of existing class is called inheritence.

Note: i) In Java extends keyword is used to perform inheritence

ii) It provides code reusability.

iii) We can't access private members of class through inheritence.

iv) A sub class contains all the features of super class so, we should create the object of sub class.

v) Method overriding only possible through inheritence.

Syntax:

```
class A          // Super Class
{
    class B extends A //
    {
        ---
    }
}
```

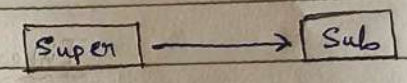
P.T.O →→

Syntax :

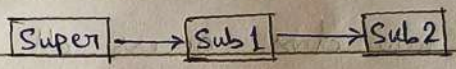
```
class A          // Super class (Parent Class)
{
  ------  +,-
  ------
}
class B extends A   // Sub Class (Child Class)
{
  ------
  ------  +,-,*,/,%
}
```

Types :

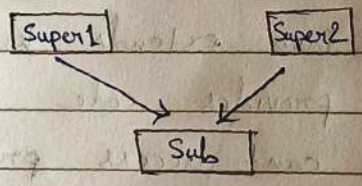
i) Single / Simple inheritence



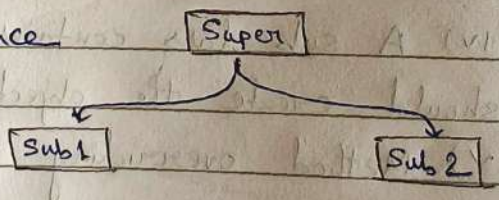

ii) Multi-level inheritence




iii) Multiple inheritence (Java doesn't support)




iv) Hierarchical inheritence




1. Simple Inheritence :

Simple inheritence is nothing but which contain only one super class and only one sub class is called Simple Inheritence.

Syntax :

```
class Super
{
  ------
}
class Sub extends Super {
  ------
}
```

```
class student    // Super Class
{
    int roll, marks;
    String name;
    void input()
    {
        System.out.println("Enter roll, name & marks: ");
    }
}
class anindya extends student    // Sub Class
{
    void disp()
    {
        roll = 1; name = "ani"; marks = 96;
        System.out.println(roll + " " + name + " " + marks);
    }
    public static void main(String[] args)
    {
        anindya r = new anindya();
        r.input();
        r.disp();
    }
}
```

→ Always try to create object from sub class because sub class contains additional features than super class.

O/P

```
Enter roll, name & marks:
1 ani 96
```

If you create a private method in super class & try to have it's features in sub class by creating an object it'll show an error bcz private methods are only accessible within it's own class where it is defined.

Instead of 'private' if we use 'protected' keyword it can be accessed with it's own class as well as the sub class that extends it. It'll run perfectly without any error.

2. Multi-Level Inheritence :

In multi-level inheritence, we have one super class and multiple sub classes called Multi-Level Inheritence.

Syntex :
```
class super
{
  ----
}
class sub1 extends super
{
  ----
}
class sub2 extends sub1
{
  ----
}
```

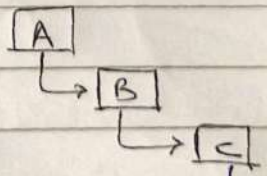
---

```
class A        // Super Class
{
  int a,b,c;
  void add()
  {
    a = 10 , b = 20;
    c = a+b;
    System.out.println("Sum is : " + c);
  }
  void sub()
  {
    a = 200 , b = 150;
    c = a-b
    System.out.println("Sub is : " +c);
  }
}
```

```
class B extends A   // Sub Class 1
{
    void mul()
    {
        a = 10, b = 3;   c = a*b;
        System.out.println("Mul is : " +c);
    }
    void div()
    {
        a = 15, b = 3;   c = a/b;
        System.out.println("Div is : " +c)
    }
}
class C extends B   // Sub Class 2
{
    void rem()
    {
        a = 10, b = 3;
        c = a%b;
        System.out.println("Remainder is : " +c);
    }
}
class Test
{
    public static void main(String[] args)
    {
        C r = new C();
        r.add(); r.sub(); r.mul(); r.div(); r.rem();
    }
}
```

Output

```
Sum is 30
Sub is 50
Mul is 30
Div is 5
Remainder is 1
```




'C' class with most features

3. Multiple Inheritence :

Whenever a sub class wants to inherit the property of two more super class that have same method, Java compiler can't decide which method it should inherit. Then there might a chance of memory duplication i.e. a reason Java doesn't support multiple inheritence through classes. But it can be implemented through interfeces which we'll talk later.

4. Hierarchical Inheritance :

An inheritance which contains only one super class and multiple sub class and all sub classes directly extend the only super class is called Hierarchical Inheritance.

Syntax :
```
class A
{
  ----
}
```

```
class B extends A
{
  ----
}
```

```
class C extends A
{
  ----
}
```

```
class A
{
  void input()
  {
    System.out.println("Enter your name : ");
  }
}
```

```
class B extends A
{
   void show()
   {
      System.out.println("My name is Anindya");
   }
}
class C extends A
{
   void display()
   {
      System.out.println("My name is Anurag");
   }
}
class Demo
{
   public Static void main(String[] args)
   {
      B r1 = new B();
      C r2 = new C();
      r1.input();  r1.show();
      r2.input();  r2.display();
   }
}
```

Output

```
Enter your name :
My name is Anindya
Enter your name :
My name is Anurag
```

# Super Keyword

Super Keyword refers to the objects of super class, it is used when we want to call the super class variable, method and constructor using sub class object.

Note :

1) Whenever the super and sub class variable and method name both are same then it can be used only.
11) To avoid the confusion b/w super and sub class variable and methods that have same name we should use super keyword.

Super Keyword
- i) Variable
- ii) Method
- iii) Constructor

```
class A
{
   int a = 10;
}
class B extends A
{
   int a = 20;
   void show()
   {
      System.out.println(a);
      System.out.println(super.a);
   }
}
class test
{
   public Static void main(String[] args)
   {
      B r = new B();
      r.show();
   }
```

Output
20
10

```
class A
{
  void show()
  {
    System.out.println("Hello viewer");
  }
}
class B extends A
{
  void show()
  {
    super.show();
    System.out.println("Hello learner");
  }
}
class test
{
  public static void main(String[] args)
  {
    B r = new B();
    r.show();
  }
}
```

Output

Hello viewer
Hello learner

---

```
class A
{
  A(int a)
  {
    System.out.println("Hello viewer" + a);
  }
}
```

```
class B extends A
{
    B()
    {
        super(100);
        System.out.println("Hello learner");
    }
}

class test
{
    public static void main(String[] args)
    {
        B r = new B();
    }
}
```

Output

Hello viewer 100

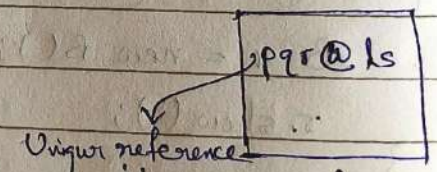
Hello learner

## This Keyword

'this' keyword refers to the current object inside a method or constructor.

example:

```
class A
{
---
---
}
A r = new A();
```




Accessed by 'r' object as well as 'this' keyword

program:

```
class A
{
    void show()
    {
        System.out.println(this);
    }
    public static void main(String[] args)
    {
        A r = new A();
        System.out.println(r); r.show();
    }
}
```

Output

A@7a81197d

A@7a81197d

Whenever the name of instance and local variables both are same then our runtime environment JVM gets confused that which one is local variable and which one is instance variable, to avoid this problem we should use 'this' keyword.

```
class A
{
    int a ;     // instance variable
    A(int a)  -> local variable
    {
        a = a ;
    }
    void show()
    {
        System.out.println(a);
    }
    public static void main(String[] args)
    {
        A r = new A();
        r.show();
    }
}
```

Output

0

Even after passing 100 as parameter value, it prints the default value of an int bez JVM is confused which one is local & which is instance

Before

```
class A
{
    int a ;
    A(int a)
    {
        this.a = a ;
    }
    void show()
    {
        System.out.println(a);
    }
}
```

this.a → instance variable coz this refers to parent obj & instance variable is part of it!

Output

100

It is also used when we want to call the default constructor of its own class.

```
class A
{
    A()
    {
      ---
      ---
    }
    A(int a)
    {
      this();
    }
}
```

it'll call the default constructor under parameterized constructor

```
class A
{
  A()
  {
    System.out.println("Learn Coding");
  }
  A(int a)
  {
    this();
    System.out.println(a)
  }
  public Static void main (String[] args)
  {
    A r = new A(100);
  }
```

Output

Learn Coding
100

It also calls parameterized constructor of its own class.

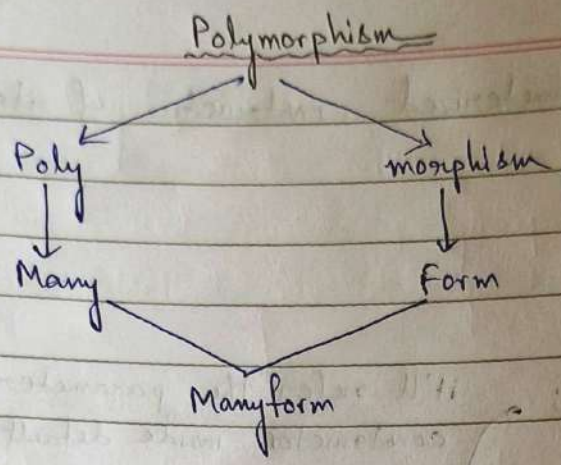
```
class A
{
    A()
    {
        this(10);
    }
    A(int a)
    {
        ==
    }
}
```

it'll refer the parameterized constructor inside default constructor

```
class A
{
    A()
    {
        this(100);
    }
    A(int a)
    {
        System.out.println(a);
    }
    public static void main(String[] args)
    {
        A r = new A();
    }
}
```

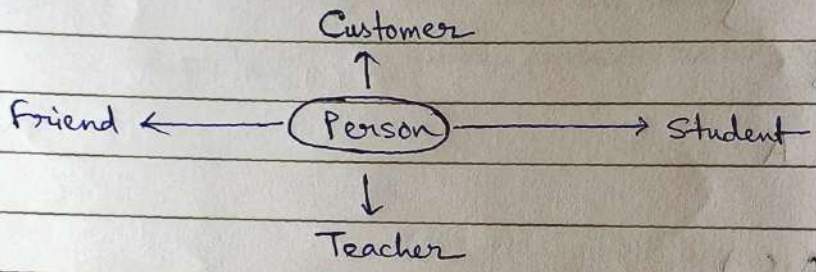
Output

100

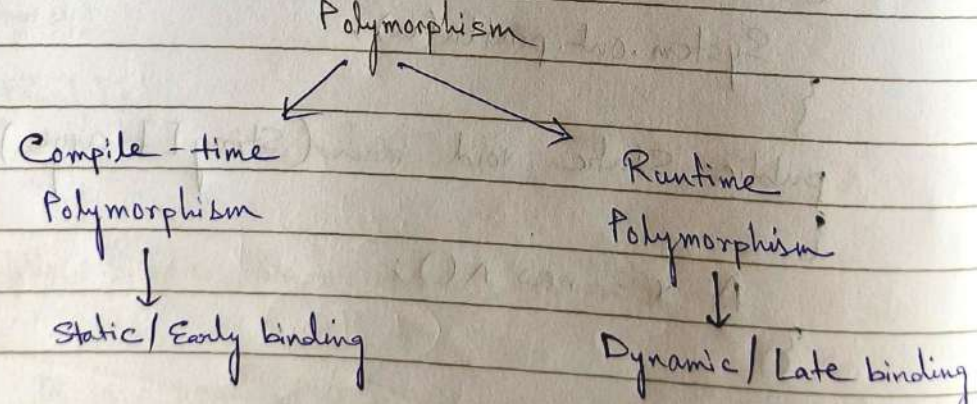



Polymorphism is the greek word whose meaning is "Same class having different behaviour".

Example:




i) void person (Teacher)
ii) void person (Student)
iii) void person (Friend)
iv) void person (Customer)



1) Compile time polymorphism

A polymorphism which exists at the time of compilation is called compile time or early binding or static polymorphism.

example: Method overloading

↓

Whenever a class contains more than one method with same name and different types of parameters called method overloading.

```
class A
{
  void add()
  {
    int a=10, b=20, c;
    c = a+b;
    System.out.println(c);
  }
  void add(int x, int y)
  {
    int ~~a=10, b=20;~~ c;
    c = ~~x~~ + ~~b~~ y;
    System.out.println(c);
  }
  void add(int x, float y)
  {
    int c;
    c = x + y;
    System.out.println(c);
  }
  public static void main(String[] args)
  {
    A r = new A();
    r.add(); r.add(100, 200); r.add(50, 45.32);
  }
```

Compiler finds out which method should be called by looking at the parameters.

Output

30
300
95.32

2) Runtime Polymorphism :

A polymorphism which exists at the time of execution of program is called runtime polymorphism.

Example : Method overriding

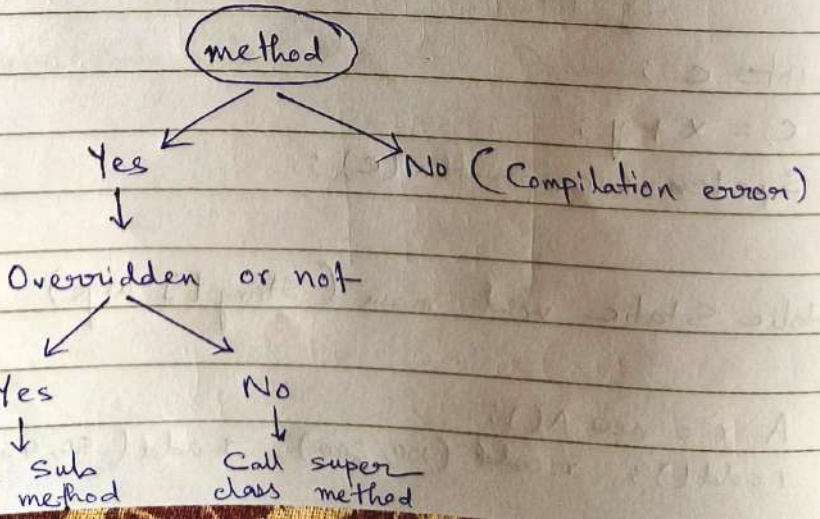
↓

Whenever we are writing method in super & sub classes in such a way that method name and param must be same called overriding.

Syntax :

```
class A
{
    void show()
    {
        ---
    }
}
class B extends A
{
    void show()
    {
        ---
    }
}
```

Method Overriding rules :



```
class shape
{
   void draw()
   {
      System.out.println("Can't say shape type");
   }
}
class square extends shape
{ @Override
   void draw()
   {
      System.out.println("square shape");
   }
}
class Demo
{
   public static void main(String[] args)
   {
      Shape r = new square();
      r.draw();
   }
}
```

Output

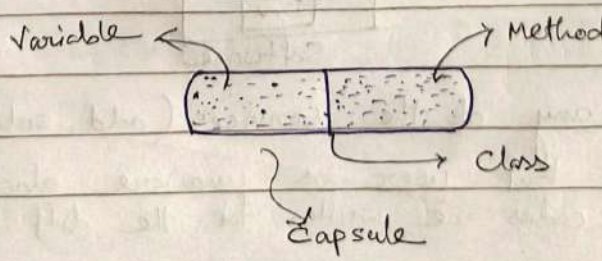
square shape

## Encapsulation

Encapsulation is a mechanism through which we can wrapping the data members and member methods of class in a single unit called encapsulation.

Note :- 1) Declare the class variables as a private.
2) Declare the class methods as a public.

Example : Class is the best example of encapsulation



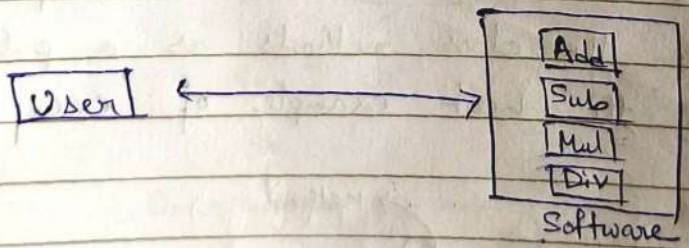
```
class A
{
    private int value; // data hiding
    public void setValue (int x)  // data abstraction
    {
        value = x;
    }
    public int getValue()
    {
        return value;
    }
}
class B
{
    public static void main (String[] args)
    {
        A r = new A();
        r.setValue(100);
        System.out.println(r.getValue());
    }
}
```

Output

100

## Abstraction

Abstraction is a process of hiding the implementation details from the user, only the highlighted ~~part~~ set of services provided to the user.



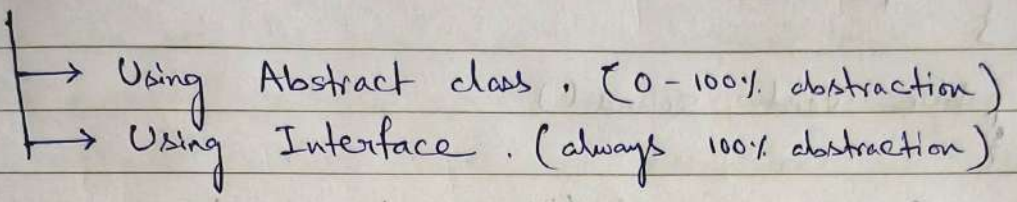

User can have any of the services (add, sub, mul, div) that software provides but user is unaware about the codes in which lang the codes are written for the diff services. This is abstraction.

Advantages :

i) Security

ii) Enhancement

Types of abstraction :




- Abstract Class :

A class which contains the abstract keyword in its declaration is called abstract class.

Note: i) We can't create object for object class

```
abstract class A
{
---
---
}
```

ii) It may or may not contain abstract methods.

iii) It can have abstract & non-abstract methods.

iv) To use an abstract class you have to inherit it from sub class.

v) If a class contain partial implementation then we should declare a class as abstract.

```
abstract class animal
{
    animal()
    {
        System.out.println("All animals...");
    }
    public abstract void sound();
}
```

```
class Dog extends animal
{
    Dog()
    {
        Super();
    }
    public void sound()
    {
        System.out.println("Dog is barking");
    }
}
class Lion extends animal
{
    Lion()
    {
        Super();
    }
    public void sound()
    {
        System.out.println("Lion is roaring");
    }
}
class Test
{
    public Static void main(String[] args)
    {
        Dog d = new Dog();
        Lion l = new Lion();
        d.sound();
        l.sound();
    }
}
```

Output

All animals
All animals
Dog is barking
Lion is roaring

Output

1. All animals
2. Dog is barking
3. All animals
4. Lion is roaring

- **Abstract Method :**

A method which contain abstract modifier at the time of declaration is called abstract method.

Note:

(i) It can only be used in the abstract class.
(ii) It doesn't contain any body "{}" and always ends with ";".
(iii) Abstract method must be overriden in sub classes otherwise it will also become a abstract class.
(iv) Whenever the action is common but the implementations are different then we should use abstract class.

```
abstract class A
{
    abstract void m1(); //abstract method
}

abstract class Programming
{
    public abstract void Developer();
}

~~abstract~~ class HTML extends Programming
{
    @Override
    public void Developer()
    {
        System.out.println("Tim Berners Lee");
    }
}

class Java extends Programming
{
    @Override
    public void Developer(){
        System.out.println("James Gosling");
    }
```

```
class Main
{
    public Static void main(String[] args)
    {
        HTML h = new HTML();
        Java j = new Java();
        h.Developer();
        j.Developer();
    }
}
```

Output

Tim Berners lee

James Gosling

## Interface

Interface is just like a class, which contains only abstract methods.

To ~~avoid~~ achieve interface Java provides a keyword called 'implements'.

Note: I) Interface methods are by default public and abstract

eg:
```
interface client
{
    void m1(); // by default public + abstract but you can also write.
}
```

II) Interface variables are by default public + static + final.

III) Interface method must be overriden in the implementing classes.

IV) Interface is nothing but a medium that deals b/w client and developer.

```
import java.util.Scanner;
interface client
{
    void input();
    void output();
}
class Developer implements client
{
    String name;
    double sal;
    @Override
    public void input()
    {
        Scanner in = new Scanner(System.in);
        System.out.println("Enter Username: ");
        name = in.nextLine();
        System.out.println("Enter salary: ");
        sal = in.nextDouble();
    }
    @Override
    public void output()
    {
        System.out.println(name + " " + sal);
    }
    public static void main(String[] args)
    {
        client c = new Developer();
        c.input();
        c.output();
    }
}
```

Output

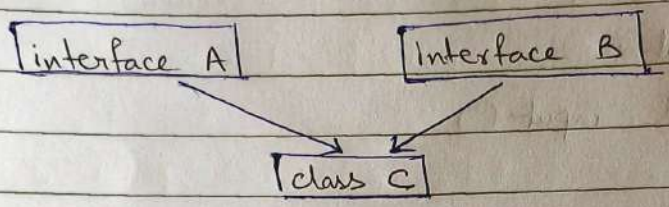
```
Enter username:
Ankit
Enter salary:
5000
Ankit 5000
```

# Multiple Inheritence Using Interface

Previously we learned that multiple inheritence isn't supported by Java compiler. But we can achieve multiple inheritence through interfaces because interface contains only abstract method, which implementation is provided by the sub class.

Note: class C extends A, B ✗

class C implements A, B ✓




```
interface A
{
   void show();
}
interface B
{
   void show();
}
class Multiple implements A,B
{
   public void show
   {
      System.out.println(" Interface A & B");
   }
   public static void main(String[] args)
   {
      Multiple m = new Multiple();
      m.show();
   }
}
```

Output

Interface A & B

# Extend Interface Using Inheritence

In inheritence we previously saw that content of a class can be reused in another class to avoid repeatative code & also time. Likewise we can also reuse the content of an inheritence into another. And it can be achieved using inheritence.

```
interface client1
{
    void add();
}
interface client2 extends client1
{
    void sub();
}
class Developer implements client2
{
    @Override
    public void add()
    {
        int a=10, b=20, c;
        c=a+b;
        System.out.println("Addition : " +c);
    }
    @Override
    public void sub()
    {
        int a=20, b=10, c;
        c=a-b;
        System.out.println("Subtraction : " +c);
    }
}
```

Output

```
Addition : 30
Subtraction : 10
```

No objects can be created for an interface but it can be used as reference.

```
public class Main {
    public static void main(String[] args) {
        client2 c = new Developer(); c.add(); c.sub();
    }
}
```