

# \* OOPS \*

11 Mar, 2020

It provides the concepts and methodologies to fulfill every project requirements.

It consists of four pillars

- ① Inheritance
- ② Polymorphism
- ③ Abstraction
- ④ Encapsulation

## ① Inheritance :-

It is the first pillar of oops concept.

- \* One class acquiring (accesing) the properties of another class is called as inheritance.
- \* The class who is acquiring the properties is called as child class (a) derived class (b) <sup>child</sup> ~~super~~ class.
- \* The class whose properties are <sup>sub</sup> acquired is called as parent class or base class (a) ~~super~~ class.
- \* Super class have its own properties.  
Sub class have its own properties and also super class.

## Extends :-

It is a keyword which indicates, we are deriving a new class from an existing class.

## Syntax :-

```
class A
{
    // methods + vars of class A //
```

```
}
class B extends A
{
    // methods & var of class A //
```

+  
// methods & var of class B //

```
}
```

Ex:-

```
class Father
{
    p v property()
    {
        S o p (" $ 's + Gold + cash ");
    }
}

class son extends Father
{
    p v glarkforleud( )
    {
        S o p (" Time waste ");
    }
}
```

## POINTS TO REMEMBER :-

⇒ In Inheritance we can have multiple classes but, basically classes are two categories

(i) Business logic class

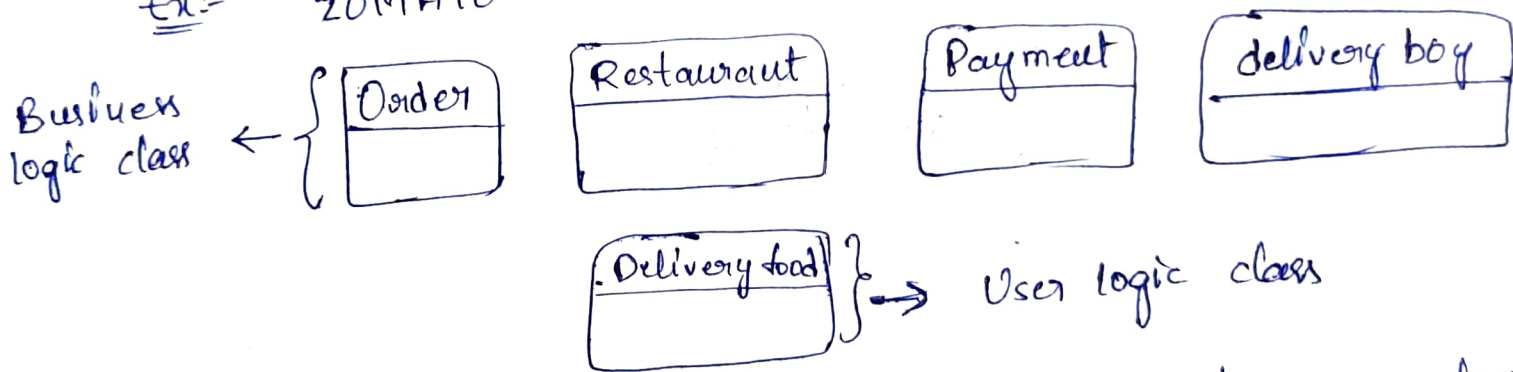
(ii) User logic class

⇒ classes which does not contains main method are called as business logic class.

⇒ Class which contains main method is called as user logic class.

⇒ It is not recommended to write a main method in business logic class rather write it in separate class.

Ex:- ZOMATO



⇒ Out of multiple classes we can keep only one class as public. and it is always recommended to keep user logic class as public.

⇒ Both static and non static methods can be inherited but only non-static methods can be overridden.

⇒ So, it is not recommended to inherit static method.

- ⇒ Constructors cannot be inherited because they are only used for initialisation of non-static variables.
- ⇒ In java we can extend only one class at a time.

Ex: X class Boy extends gfi, gfi2

- ⇒ For every class of java predefined or userdefine there is always one default superclass is there whose name is object class.

- ⇒ Object class is present in java.lang package.

Ex:-

```

class A (extends object)
{
}
    
```

→ Contains 11 methods ( )'s  
↓  
is imp

defaultly added

```

class A extends object
{
}
    
```

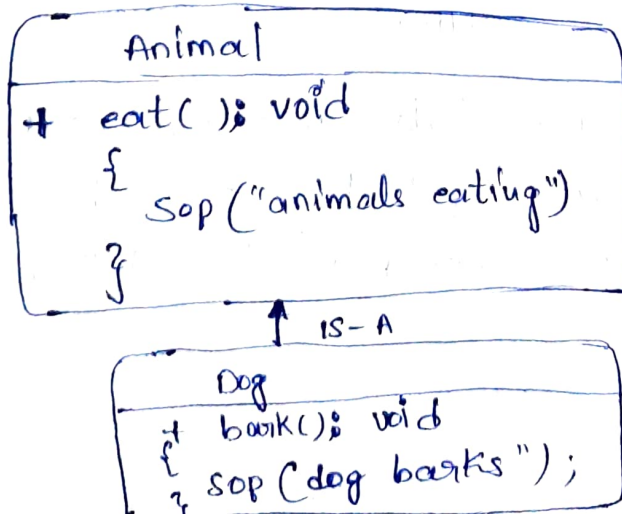
extends class A

```

class B
{
}
    
```

## TYPES OF INHERITANCE :-

### ① Single Level :-



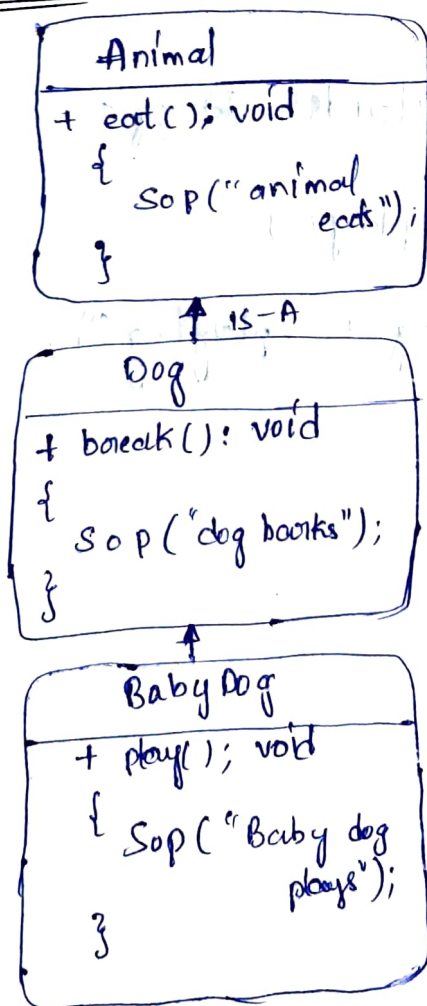
Prgm:-

```
public class DE extends object
{
    p s v m (String args[])
    {
        Dog d1 = new Dog();
        d1.bark();
        d1.eat();
    }
}

class Animal
{
    public void eat()
    {
        Sop("Animal eats");
    }
}

class Dog extends Animal
{
    public void bark()
    {
        Sop("Dog barks");
    }
}
```

② Multi level :-





Prgm:

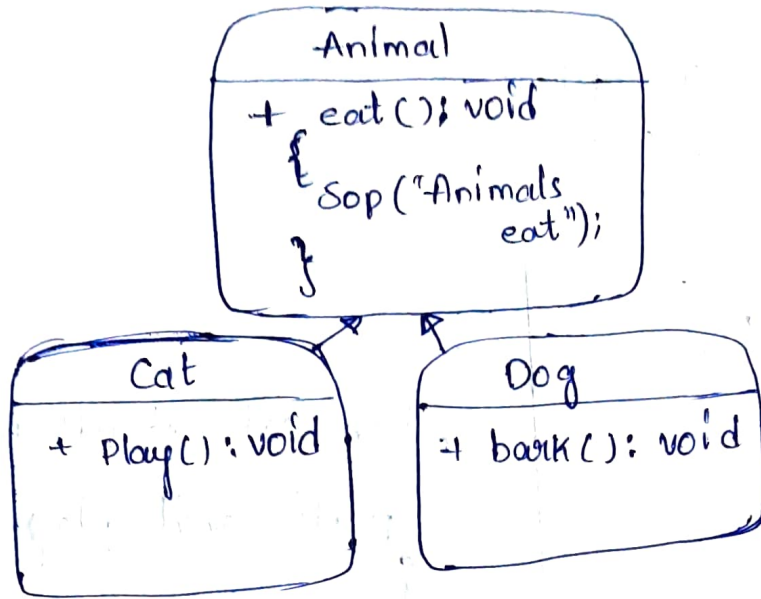
```
public class Dog
{
    public static void main(String args)
    {
        Dog d1 = new Dog();
        d1.bark();
        d1.eat();
        d1.play();
    }
}

class Animals
{
    public void eat()
    {
        System.out.println("Animal eats");
    }
}

class Doggie extends Animals
{
    public void bark()
    {
        System.out.println("Dog barks");
    }
}

class BabyDog extends Doggie
{
    public void play()
    {
        System.out.println("Baby dog is playing");
    }
}
```

## ③ Hierarchical level :-



Prgm :-

```
public class DC
{
    p s v m (String args[])
    {
        d1. eat();
        d2. play();
        d3. bark();
    }
}
```

Dog d1 = new Dog();

```
class Animals
{
    public void eat()
    {
        Sop("Animal eats");
    }
}
```

```
class cat extends Animals
```

```
{
    public void play()
```

```
{
    Sop("cat. escapes");
}
```

```
class Dog extends Animal
```

```
{
    public void bark()
```

```
{
    Sop("Dog bites");
}
```

Prgm

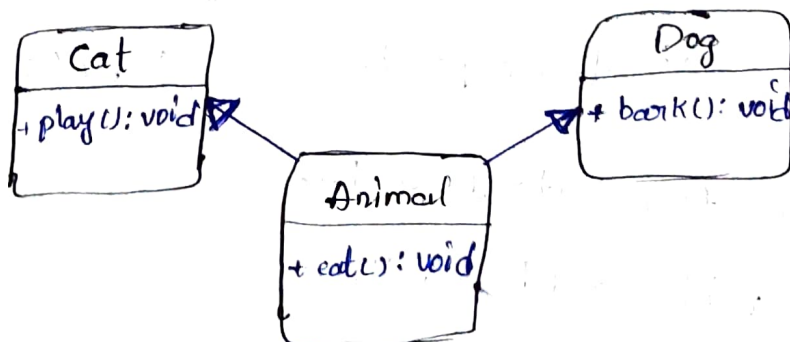
```
{ public class DE
{
    p s v m (String args[])
    Doggie d1 = new Doggie();
    d1.bark();
    d1.eat();
    cat d2 = new cat();
    d2.play();
    d1.eat();
}
}
class Animals
{
    public void eat()
    {
        S.o.p ("Animal eats");
    }
}
class Doggie extends Animals
{
    public void bark()
    {
        S.o.p ("Dog barks");
    }
}
```

```
{ class cat extends Animals
{
    public void play()
    {
        S.o.p ("cat is playing");
    }
}
```

### Multiple Inheritance:-

One class is extending (or) inheriting two super classes at the same time is called as multiple inheritance.

Ex:-





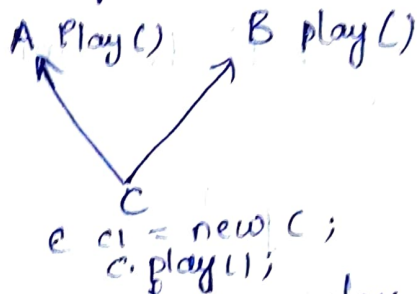
→ Multiple inheritance is not possible through classes because one class cannot extend more than one class at the same time.

Because there will be a ambiguity problem, ~~also~~ diamond problem & constructor chaining problem.

### Ambiguity problem:

As per multiple inheritance one class is extending two immediate super classes, if both the classes contains same method name during method calling JVM will get confuse which method to call.

It is called ambiguity problem.



Since the shape of a class is like a diamond structure, so it is referred as diamond problem.

### \* Super() :-

The process of calling super class constructor from subclass constructor is called as call to super.

→ Call to super must be the first statement in a constructor.

⇒ Call to super is 'implicit' and explicit computer in nature.

Super() as 'implicit':


Call to super() is 'implicit' in nature i.e., added by JVM, if super class constructor does not contain any arguments.

Ex:-

```
class Animal
{
    public Animal()
    {
        Sop("Animal constructor");
    }
}

class Dog extends Animal
{
    public Dog()
    {
        // super()
        Sop("In Dog constructor");
    }
}

class user
{
    main()
    {
        Dog d1 = new Dog();
    }
}
```



o/p:- Two statements will print i.e., both dog and animal's class

Super(); as explicit :-

=> It is explicit in nature i.e., has to be added by programmer, if a super class contains any arguments.

Ex:-

```
class Animal
{
    public Animal (String name)
    {
        Sop("Animal constructor");
    }
}
```

```
class Dog extends Animal
{
    public Dog()
```

```
    {
        Super("lion");
        Sop("dog constructor");
    }
}
```

```
class user
{
    main()
    {
        Dog d1 = new Dog();
    }
}
```

→ If we don't write explicitly, we will get CTE

# Combination of call to this(); a Super();

Ex:-

```
public class DE
```

```
{  
    p s v m (String args())  
    {  
        Doggie d1 = new Doggie();  
    }  
}
```

O/p:-  
①  
②  
③  
④

```
class Animals
```

```
{  
    public Animals() {  
        this ("Lion");  
    }  
}
```

class Doggie extends Animals

```
{  
    public Doggie() {  
        this ("Tommy");  
    }  
}
```

```
② Sop ("Anonymous animal");
```

```
{  
    public Animals (String name)
```

```
④ Sop ("anonymous dog");  
    }  
    public Doggie (String name)  
    {  
        // super() //  
    }  
}
```

```
① Sop ("Animal is " + name);
```

```
{  
}
```

```
③ Sop ("Doggie name is " + name);  
    }  
}
```

## Construction chaining problem :-

Ex:-

```
class A
```

```
{  
    public A() {  
    }  
}
```

class C extends A B

```
{  
    public C() {  
        super();  
    }  
}
```

```
class B
```

```
{  
    public B() {  
    }  
}
```

O/p CTE

From the above example we can infer that if a class extends more than one class there is a chance of constructor chaining problem.  
Therefore due to ambiguity problem, diamond problem, if constructor chaining problem multiple inheritance is not possible through classes.

### Hybrid Inheritance:-

It is a combination of multiple inheritance and hierarchical inheritance, since multiple inheritance and hybrid inheritance are not possible.

### Advantages of Inheritance:-

- \* Reusability of coding.
- \* Flexible to extend.
- \* Easy to understand.

### Working with ECLIPSE :-

Install the eclipse by referring document.

Open the application

Provide workspace (location to save a program)

### Step-1:- Creating a project

click on File → new → project → java project → provide project name → Finish



⇒ Under project explorer we can observe that one project folder get connected with the library file and source folder.

### Step 2 :- Creating a package

Right click on source folder → new → package  
→ verify source folder name and provide package name → finish

⇒ We can observe that under source folder one package got created.

### Step 3 :- Creating a class

Right click on source folder → new → class  
verify source folder name and package name.  
provide name of a class.

→ Select the access modifier

→ click on finish.

We can observe that our class will get created under given package.

⇒ Like this under single package we can create multiple classes.


## Method Overriding

During inheritance subclass has complete privilege to change the implementation of super class.

This process is called method overriding.

During method overriding superclass behaviour will be hidden and subclass behaviour is visible because in method overriding which method to call will be decided depending on run time object.

Ex:-  
parent P1 = new parent();  
child C1 = new child();



The diagram shows two boxes labeled 'parent' and 'child' with arrows pointing from them to a single box labeled 'Runtime object'. The 'parent' and 'child' boxes are enclosed in a dashed line.

⇒ We go for method overriding when we want to change the implementation of super class as per our requirement.

Ex program:-

```
public class user
```

```
{  
    String args[]
```

```
{  
    My m1 = new My();
```

```
    m1.ringtones();  
}
```

```
}  
class people
```

```
{  
    public void ringtones()
```

```
{  
        Sop ("corona");  
    }
```

```
}
```

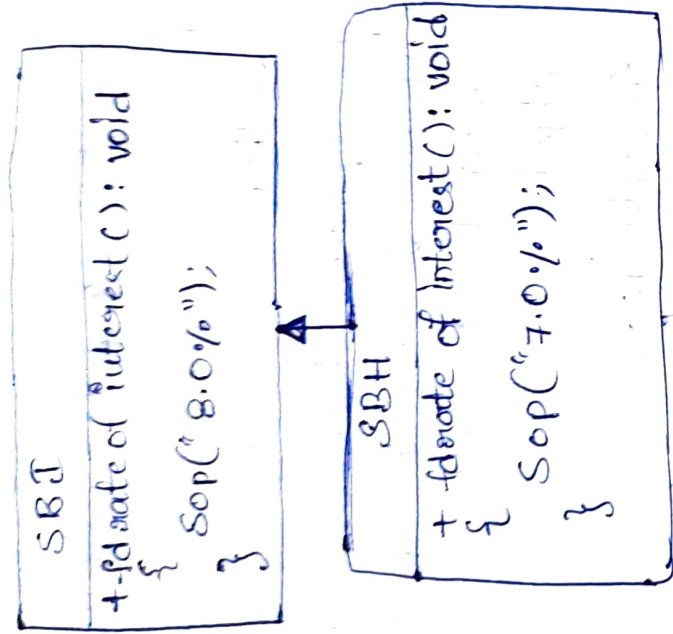
```
class my extends people
```

```
{  
    public void ringtones()
```

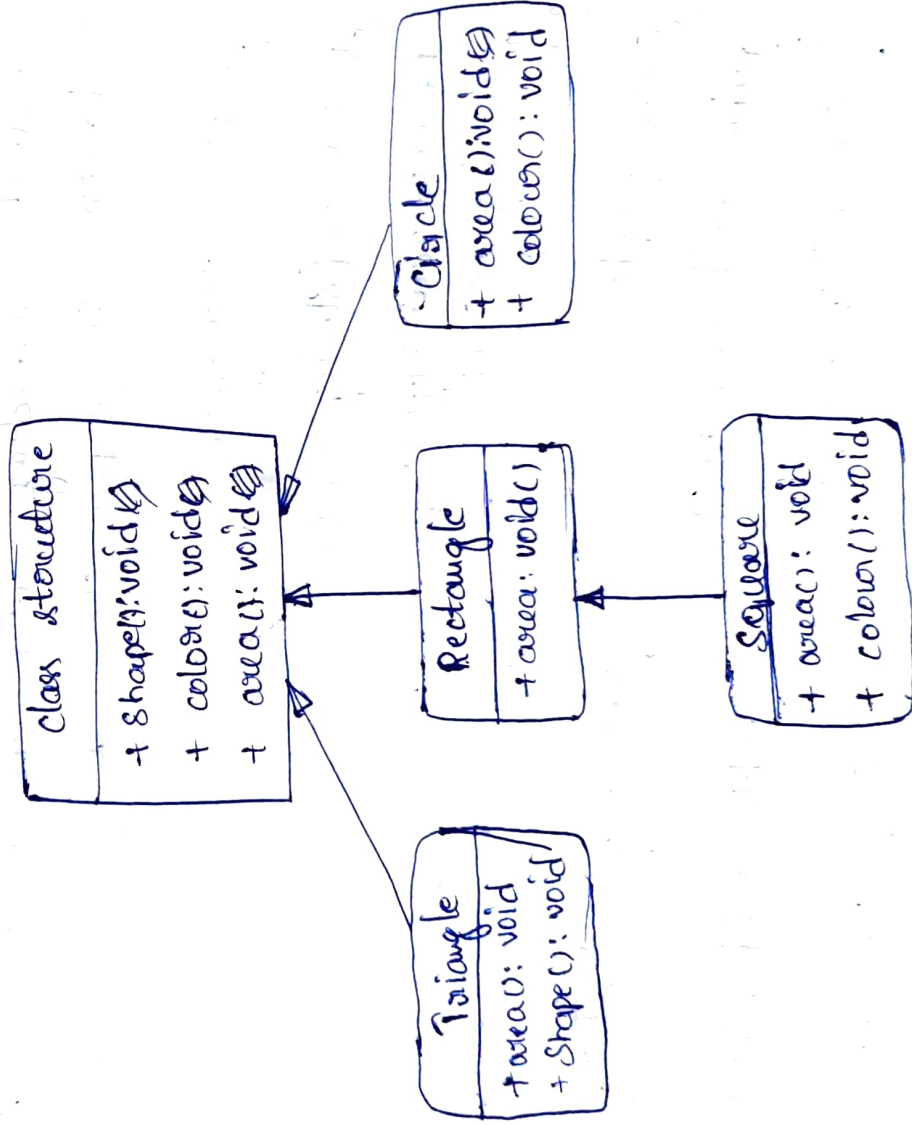
```
{  
        Sop ("tin tin...");  
    }
```

```
}
```

1



2



①

Program:-

```
public class main
{
    p s v m(storing args[3])
    {
        SBH s1 = new SBH();
        s1-fdmate of interest ();
    }
    class SBH
    {
        public void fdmateofinterest()
        {
            sop("8.0%");
        }
    }
}
```

```
class SBH extends SBH
{
    public void fdmateofinterest()
    {
        sop("7.0%");
    }
}
```

②

```
public class main
{
```

### Rules for method overriding :-

- ⇒ Inheritance is mandatory.
- ⇒ In superclass and subclass method signature must be same.
- ⇒ Return type must be same.
- ⇒ Super class method should not be private.
- ⇒ Super class method should not be final.
- ⇒ Because final methods cannot be overridden.
- ⇒ Superclass methods should not be static because if you try to override static method it leads to method hiding concept.