Inheritance

* It is process of acquiring features of an existing class into a new class.
* The class that inherits properties is called the subclass or derived class or child class.
* And the class that provides properties is called the super class or base class or parent class.
* In java, "extends" keyword is used to establish an inheritance relationship b/w two classes.

Example -

→ A cylinder can acquir all the properties of circle plus it can have extra features, where we can write a class cylinder inherting from class circle.

```
Class Circle   // Parent class
{
  public double radius;

  public double area()
   {
      return Math.PI * radius * radius;
   }

  public double perimeter()
  {
     return 2 * Math.PI * radius;
  }

  Public double circumference()
   {
      return perimeter();
   }
}
```

```
class Cylinder extends Circle  // Derived class
{
    public double height;

    public double volume()
    {
        return area() * height;
    }
}

Public class Test
{
    Public static void main(String a[])
    {
        Cylinder c = new Cylinder();
            c.radius = 7;
            c.height = 10;

        S.o.p("Volume: "+ c.volume());

        S.o.p("Area: "+ c.area());
    }
}
```

area() method is not created in cylinder class but still we can find the area of cylinder because we inheriting the property of circle class.

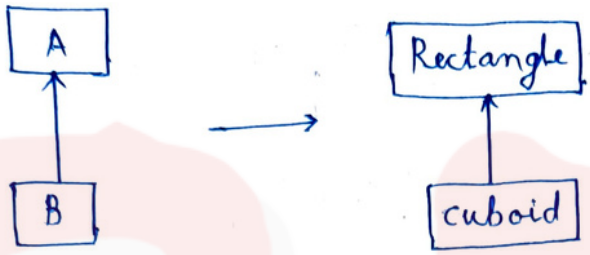
Note:

- Circle and cylinder are the two classes where circle class is having radius and cylinder class acquire the property (radius) of the circle with an extra property that is height

* Types of Inheritance -

i) Single Inheritance -

When a class is inherited from an existing class.

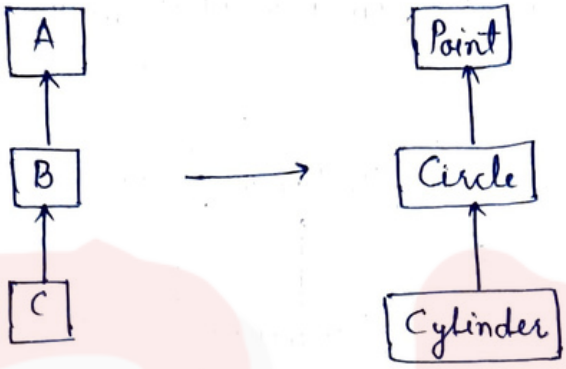


Eg:-
```
Class Rectangle
{
    int length;
    int breadth;

    Public int area()
    {
        return length * breadth;
    }
}
Class Cuboid extends Rectangle
{
    int height;
    Public void volume()
    {
        return length * breadth * height;
    }
}
Public Class Test
{
    P.S.V.m(String[] args)
    {
        Cuboid c = new Cuboid();
        c.length = 5; c.breadth = 4; c.height = 3;
        S.o.p("Area of rectangle:" + c.area())
        S.o.p("Volume of Cuboid:" + c.volume());
    }
}
```

ii) Multilevel Inheritance -

When each class are inherited from one another.



Eg:-

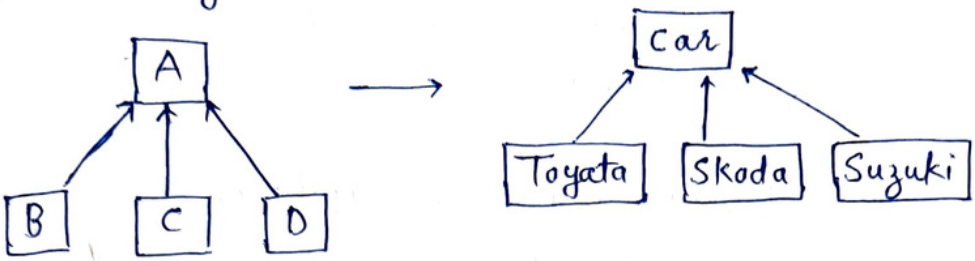
```
Class Point  // Parent class
{   int x;
    int y;
    Public Point (int x, int y)
    {   this.x = x;
        this.y = y;
    }
}

Class Circle extends Point // Child class inheriting from point class
{   int radius;
    Public Circle(int x, int y, int radius)
    {   super (x, y);
        this. radius = radius;
    }
}

Class Cylinder extends Circle // Child class inheriting from Circle class
{   int height;
    Public Cylinder(int x, int y, int radius, int height)
    {   Super (x, y, radius);
        this. height = height;
    }
}
```

iii) Hierarchical Inheritance -

When more than classes are inherited from a single class.




```
class Car
{
   void drive()
    {
      S.o.P("Driving a car");
    }
}
class Toyota extends Car
 {
    void hybridDrive()
     {
        S.o.P("Driving a Toyota car in hybrid mode");
     }
 }
class Skodo extends Car
  {
     void sportDrive()
      {
        s.o.P("Driving a skoda car in sport mode");
      }
  }
class Suzuki extends Car
   {
     void automaticDrive()
      {
        S.o.P("Driving a Suzuki car in automatic mode");
      }
   }
```

```
Public class CarTest
{
    Public static void main(String a[])
    {
        Car car1 = new Car();
        car1.drive();

        Toyota car2 = new Toyota();
        car2.drive();
        car2.hybridDrive();

        Skoda car3 = new Skoda();
        car3.drive();
        car3.sportDrive();

        Suzuki car4 = new Suzuki();
        car4.drive()
        car4.automaticDrive();
    }
}
```

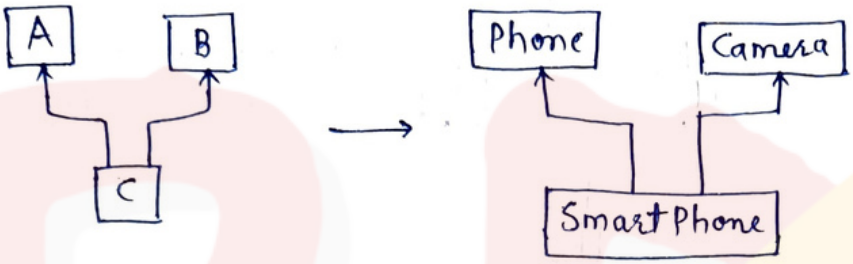
Note:

In this example the class 'car' serves as the base class or superclass while the classes 'Toyota', 'Skoda' and 'Suzuki' are the subclasses that inherited from 'car'.

iv) Multiple Inheritance -

When a class are inherited from more than one class.

→ It means, for one class, there can be more than one base classes.



* But in java, a class can extends only one class at a time.

So multiple inheritance is not possible classes because of -

i) Ambiguity -

If one class extends two classes and in case, if both classes contains same method then while calling a method, JVM will get confuse which class method to call this problem, is known as Ambiguity problems.

ii) Diamond Problem

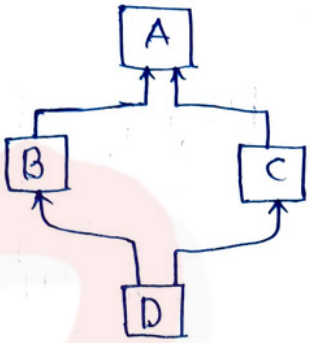
iii) Constructor chaining problem

Note :

We can achieve multiple inheritance in java using 'Interfaces'.

v) Hybrid Inheritance -

When more than one inheritance are mixed each other then it is known as hybrid inheritance.



(Combination of hierarchical and multiple inheritance)

* It is not directly supported in java only through interfaces we can achieve this.

* Important Points-

→ We can inherit public, protected and defaut methods.

→ We can not inherit private method because they are accessible only within class.

→ We can inherit non-static, final and abstract methods.

→ We can not inherit static methods because they will loaded only once in SPA.

→ We can not inherit constructors because they are mainly used for initialisation and they are not member of class.

* isA v/s has A relation -

(is A)

(has A)

```
class Rectangle
{
  ---
  ---
}
```

```
class Cuboid extends Rectangle
{
  ----
  ---
  ---
}
```

```
class Table
{
  Rectangle top;
  ---
  ---
}
```

→ (This class is inheriting from Rectangle, so we can say cuboid is A Rectangle)

→ So the relation b/w Rectangle class and class is isA

→ This class having a object of rectangle class, so we can say, Table class has A Rectangle

* Constructor in inheritance -

→ Constructors are the methods of class which are automatically called when an object is created.

→ Constructors are executed from top to bottom.

→ When we create an obj of child class, then the constructor of parent class is executed first, then the constructor of child class is executed.

Eg:-

```
class Parent
{
    Public Parent()  // Parent constructor
    {
        S.o.p(" Parent Constructor");
    }
}

class child extends Parent
{
    Public child()  // Child constructor
    {
        S.o.p(" Child constructor");
    }
}

class grandchild extends child
{
    Public grandchild()  // grand child constructor
    {
        S.o.p(" grand child constructor");
    }
}
```

```
public class Test
{
   Public static void main (String a[])
    {
        grandChild g = new grandchild ();
    }
}
```

O/P - Parent constructor
child constructor
grandchild constructor

→ In the given example program, if a grandchild class object is created then as we know grandchild class inherits child class and child class inherits parent class, so the grandchild objects first executes the top most class then level by level.

* 'super' keyword -

The 'super' keyword is used to refer to the parent class or super class of the current class.

It can be used in several contexts.

1. Call a superclass constructor
2. Call a superclass method.
3. Accessing the variables of the parent class.

* Use of 'super' keyword in calling a superclass constructor -

```
class Parent
{
   Parent()
   {
      S.o.p("Non-parameterized constructor of Parent");
   }
   Parent(int x)
   {
      S.o.p("Parameterized constructor of Parent"+x);
   }
}
class Child extends Parent
 {
    child()
    {
       S.o.p("Non-param of child");
    }
    child(int y)
    {
       S.o.p("Param of child");
    }
```

```
child (int x, int y)
{
    super(x); // It will call parameterized constructor of parent class.
    S.o.p("two param of child "+y);
}

Public class Test
{
    Public static void main(String[] args)
    {
        child c = new child();
        child c = new child(20);
        child c = new child(10, 20);
    }
}
```

→ It must be called from the child class constructor

→ This line must be a first line

o/p - Non-Param of Parent
Non-Param of child
} new child()

Non-param of Parent
~~Parent~~ Param of child
} new child(20)

Param of Parent 10
two param of child 20
} new child(10,20)

Example-2 (Accessing variables and methods)

```
Class Vehicle
{
   int speed = 50;
   void message():
   {
      S.o.P ("Welcome to vehicle class");
   }
}
Class Bike extends Vehicle
{
   int speed = 100;
   void message()
   {
      S.o.P (" Welcome to Bike class");
   }
   void display()
   {
      S.o.P(" Bike speed is :" + speed); //calling the 'Bike' class variable
      S.o.P(" Vehicle Avg speed is :" + super.speed);
      message();
      super.message();
   }
}
Public class Test
{
   Public static void main (String[] args)
   {
      Bike b = new Bike();
      b.display()
   }
}
```

It will call 'Bike' class method

Calling the 'Vehicle' class variable

It will call 'Vehicle' class method

O/P - Bike speed is : 100
Vehicle Avg speed is: 50
Welcome to Bike class.
Welcome to Vehicle class.