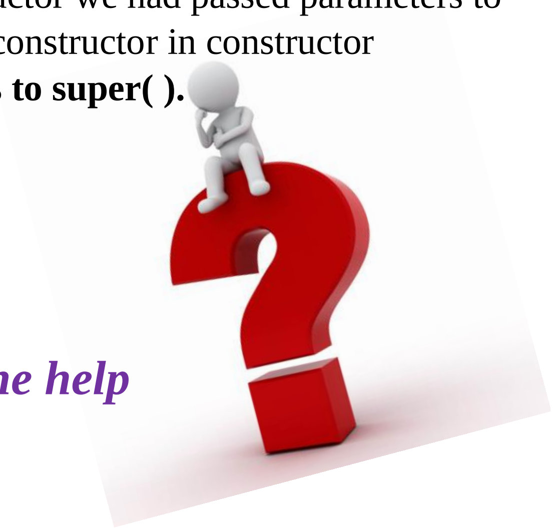
Calling parameterized constructor.

In the previous example we had seen constructor chaining of zero parameterized constructor using super() call.

Calling parameterized constructor in constructor chaining is similar to Calling parameterized constructor in local chaining.

In local chaining, to call parameterized constructor we had passed parameters to **this( ),** in the same way to call parameterized constructor in constructor chaining all we have to do is **pass parameters to super( ).**



Let us try to understand this with the help Of code:

```
class Object
{
        Object( )
        {

        }
}
class Test1 extends Object
{
    int x,y;
    Test1() //zero parameterized constructor
    {
        x=100;
        y=200;
    }
    Test1(int x, int y) //parameterized constructor
    {
        super();
        this.x = x;
        this.y = y;
    }
}
```

```
class Test2 extends Test1
{
    int a,b;
    Test2() //zero parameterized constructor
    {
        a=300;
        b=400;
    }
    Test2(int a, int b) //parameterized constructor
    {
        super(a,b);
        this.a = a;
        this.b = b;
    }
    void disp()
    {
        System.out.println(x);
        System.out.println(y);
        System.out.println(a);
        System.out.println(b);
    }
}
class Demo
{
    public static void main(String[ ] args)
    {
        Test2 t2 = new Test2(9,99);
        t2.disp();
    }
}
```

Call to parameterized constructor

OUTPUT

```
9
99
9
99
```

After learning local chaining and constructor chaining in detail, try to predict the output of the code given below which consists of both local and constructor chaining.

Exercise:

```
class Object
{
        Object( )
        {

        }
}
```

```
class Test1 extends Object
{
    int x,y;
    Test1() //zero parameterized constructor
    {
        super(); //call to object class constructor(constructor chaining)
        x=100;
        y=200;
    }
    Test1(int x, int y) //parameterized constructor
    {
        this.x = x;
        this.y = y;
    }
}
class Test2 extends Test1
{
    int a,b;
    Test2() //zero parameterized constructor
    {
        this(9,99); //call to pararmeterized constructor in same class(local chaining)
    }
    Test2(int a, int b) //parameterized constructor
    {
        super(); //call to parent class constructor(constructor chaining);
        this.a = a;
        this.b = b;
    }
```

```
      void disp()
      {
            System.out.println(x);
            System.out.println(y);
            System.out.println(a);
            System.out.println(b);
      }
}
class Demo
{
      public static void main(String[ ] args)
      {
            Test2 t2 = new Test2();
            t2.disp();
      }
}
```

OUTPUT

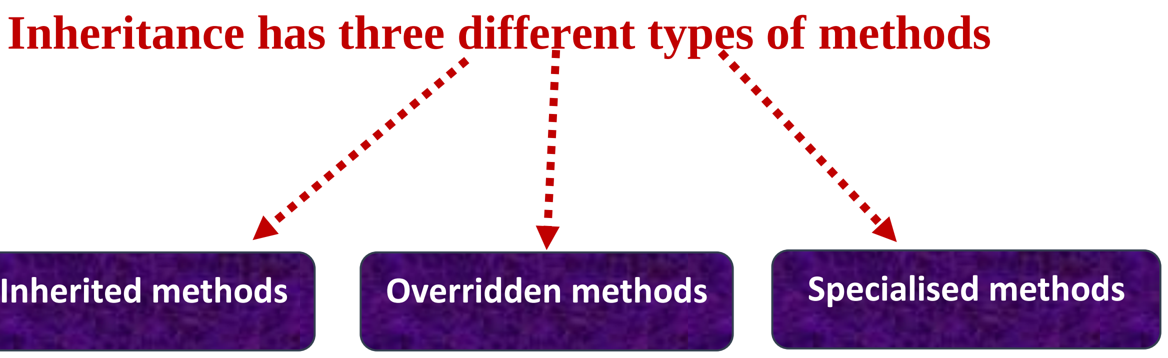
```
100
200
9
99
```

He got the correct output. Did you?

Key points:

- The first line of a constructor can be either the super() call or this() call.
- Super() call would perform **constructor chaining**, whereas this() call would perform **local chaining.**
- Both this() and super() must compulsorily be the **first line of constructor**, hence the first line of constructor can only be either this() or super() but not both.

Types of Methods in inheritance



A derived class has the ability to inherit, redefine/override an inherited method, replacing the inherited method by one that is specifically designed for the derived class and to have a method which is not present in base class.

Definitions:

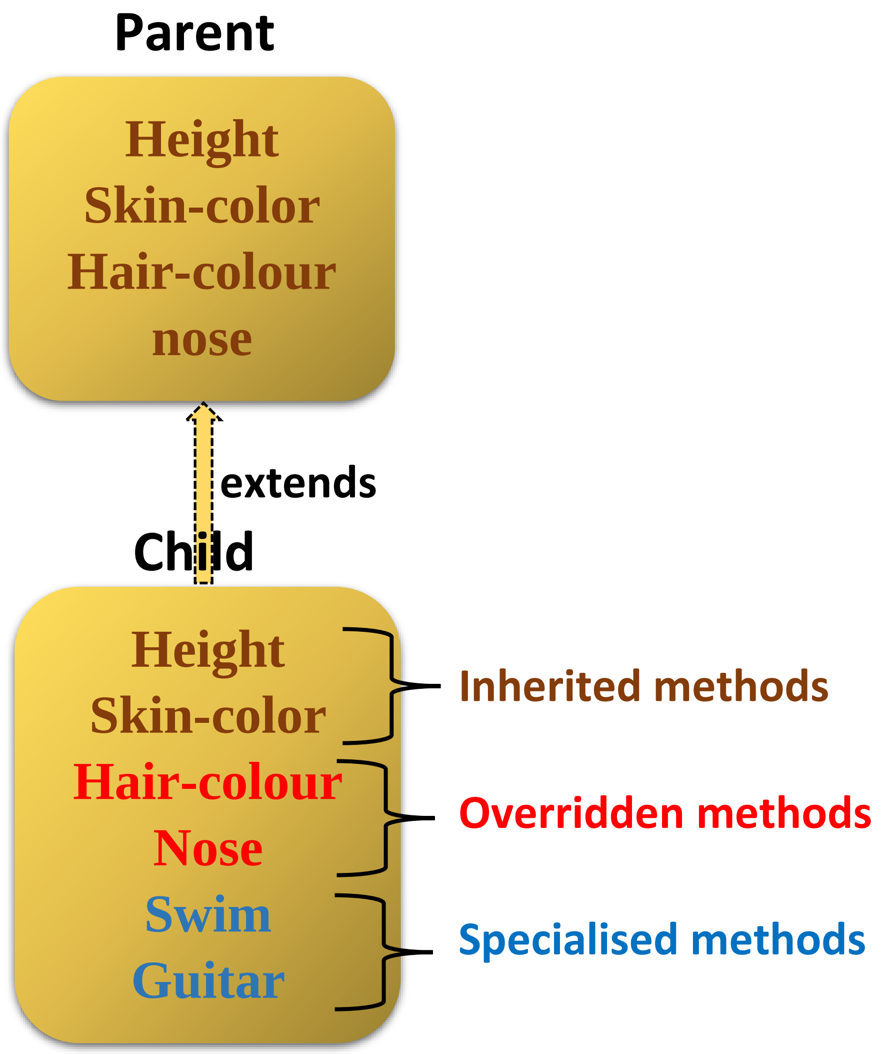
Inherited Methods are such methods which are inherited from parent class and used without any modification in child class.

Overridden methods are such methods which are inherited from parent class and are modified and used in child class.

Specialised methods are such methods which are only present in child class but not in parent class.

Let us try to understand these methods in detail with the help of Parent-Child example.

Consider the UML diagram shown below:



In the above diagram, height and colour are inherited from the parent and used without modification, hence they are inherited methods.
Hair-colour is inherited and modified by the child; hence it is now an overridden method. Similarly, nose is modified and used by the child and hence it is also an overridden method.
Swim and Guitar are not present in parent class but in child class, hence they are specialised methods.

Let us start coding to understand these methods.

```
class Plane
{
    void takeoff()
    {
        System.out.println("Plane is taking off");
    }
    void fly()
    {
        System.out.println("Plane is flying");
    }
    void land()
    {
        System.out.println("Plane is landing");
    }
}
class CargoPlane extends Plane
{
    void fly()
    {
        System.out.println("Plane is flying at low heights");
    }
}

class PassengerPlane extends Plane
{
    void fly()
    {
        System.out.println("Plane is flying at medium heights");
    }
}
class FighterPlane extends Plane
{
    void fly()
    {
        System.out.println("Plane is flying at higher heights");
    }
}
```

```
class Demo
{
    public static void main(String[ ] args)
    {
        CargoPlane cp = new CargoPlane();
        PassengerPlane pp = new PassengerPlane();
        FighterPlane fp = new FighterPlane();

        cp.takeoff();
        cp.fly();
        cp.land();

        pp.takeoff();
        pp.fly();
        pp.land();

        fp.takeoff();
        fp.fly();
        fp.land();
    }
}
```

OUTPUT:

```
Plane is taking off
Plane is flying at low heights
Plane is landing
Plane is taking off
Plane is flying at medium heights
Plane is landing
Plane is taking off
Plane is flying at higher heights
Plane is landing
```

If you carefully look at the above output, it's the overridden methods that gets executed not the inherited methods.