# OBJECT ORIENTED PROGRAMMING USING JAVA

## OUTLINE

- Method Overriding
- Super Keyword
- Types of Inheritance

## METHOD OVERRIDING

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
- In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

❑ **Usage of Overriding**

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism.

## RULES FOR METHOD OVERRIDING

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

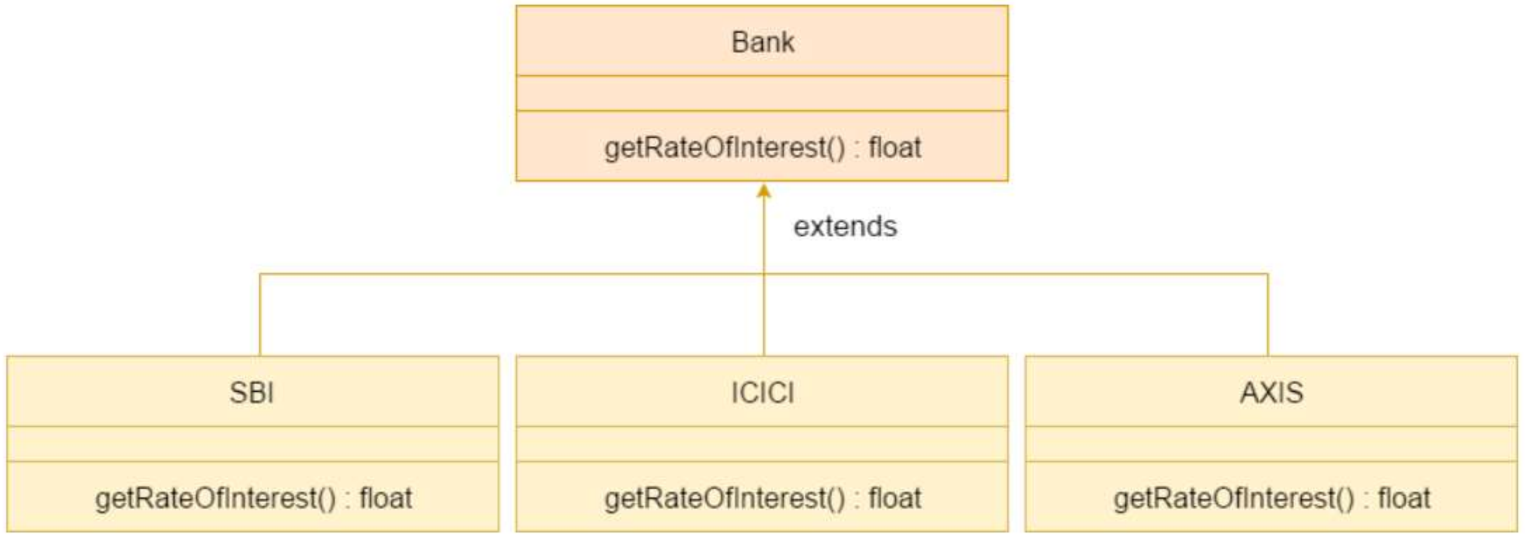
## EXAMPLE OF METHOD OVERRIDING

```
//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){
   System.out.println("Vehicle is running");
  }
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){
     System.out.println("Bike is running safely");
}

  public static void main(String args[]){
  Bike2 obj = new Bike2(); //creating object
  obj.run(); //calling method
  }
}
```

**Output:**
**Bike is Running Safely**

## EXAMPLE OF METHOD OVERRIDING

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.



Bank
getRateOfInterest() : float
extends
SBI
getRateOfInterest() : float
ICICI
getRateOfInterest() : float
AXIS
getRateOfInterest() : float

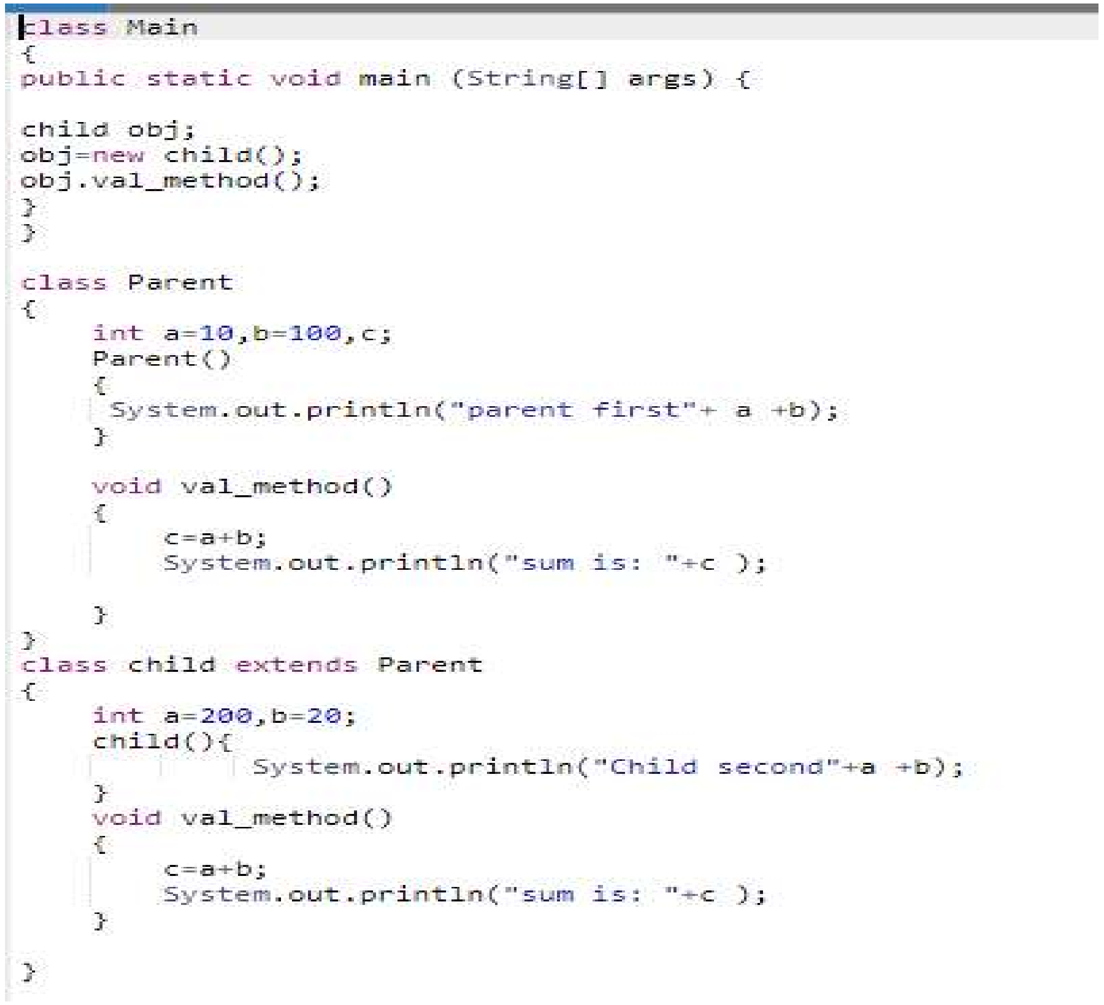
## A REAL EXAMPLE OF METHOD OVERRIDING

```
//Creating a parent class.
class Bank{
int getRateOfInterest(){return 0;}}
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest(){return 8;}  }
class ICICI extends Bank{
int getRateOfInterest(){return 7;}  }
class AXIS extends Bank{
int getRateOfInterest(){return 9;}  }
//Test class to create objects and call the methods
class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}  }
```

Output:

```
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

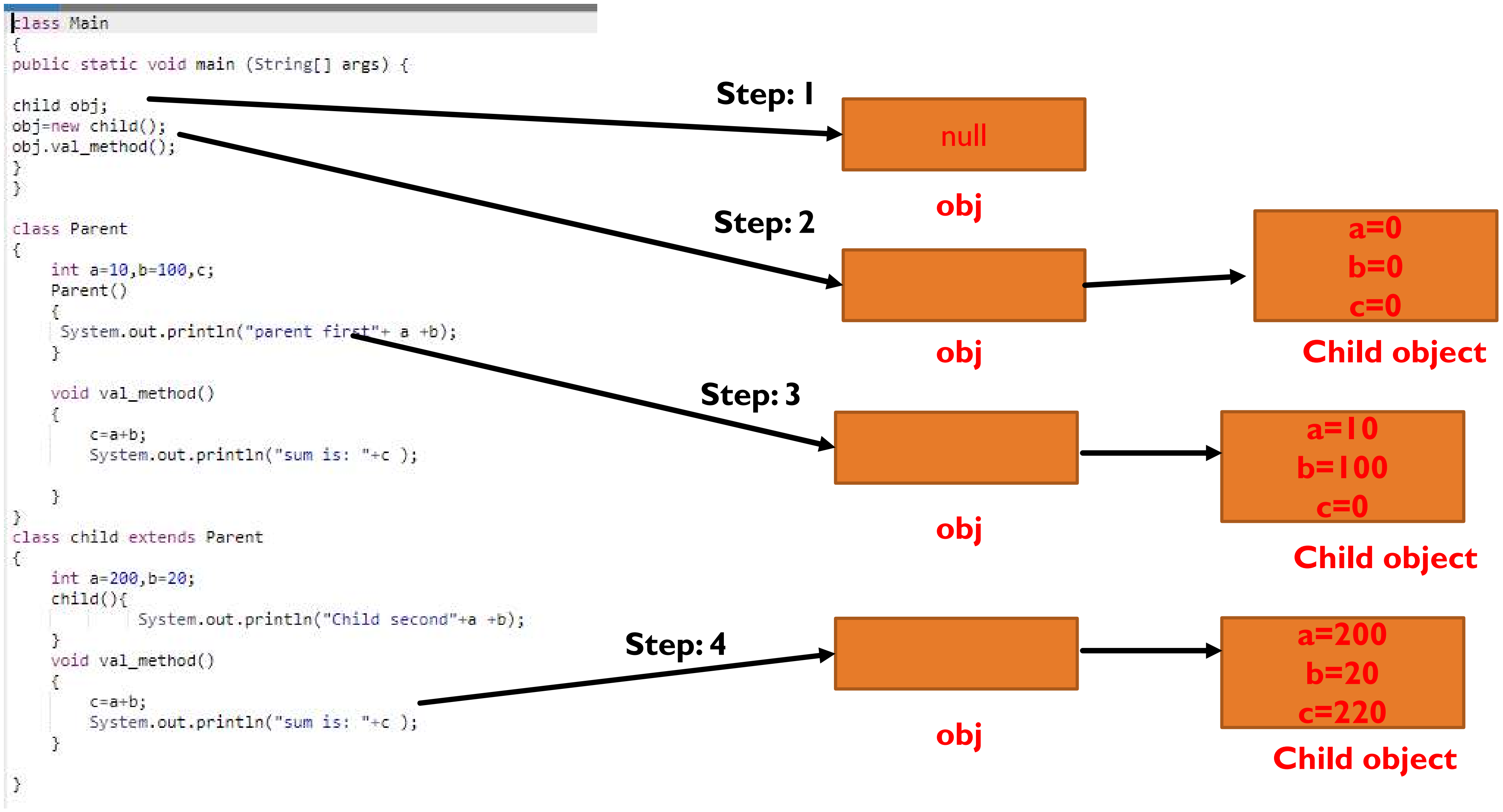
## PROBLEM DESCRIPTION: NEED OF SUPER KEYWORD



```
class Main
{
public static void main (String[] args) {

child obj;
obj=new child();
obj.val_method();
}
}

class Parent
{
    int a=10,b=100,c;
    Parent()
    {
     System.out.println("parent first"+ a +b);
    }

    void val_method()
    {
        c=a+b;
        System.out.println("sum is: "+c );

    }
}
class child extends Parent
{
    int a=200,b=20;
    child(){
            System.out.println("Child second"+a +b);
    }
    void val_method()
    {
        c=a+b;
        System.out.println("sum is: "+c );
    }

}
```

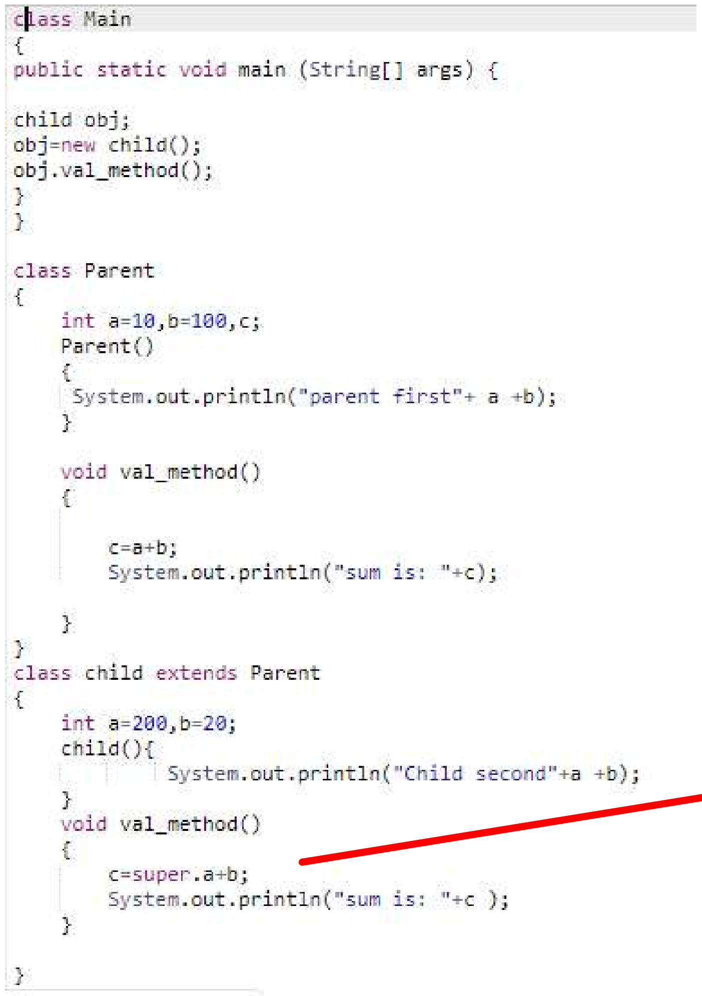
**Output:**
**Parent first10100**
**Child second20020**
**Sum is 220**

## PROBLEM DESCRIPTION: NEED OF SUPER KEYWORD



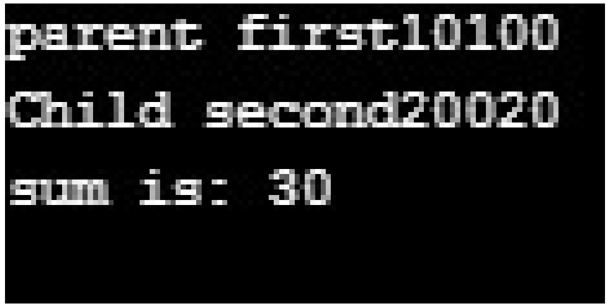
```
class Main
{
public static void main (String[] args) {

child obj;
obj=new child();
obj.val_method();
}
}

class Parent
{
    int a=10,b=100,c;
    Parent()
    {
     System.out.println("parent first"+ a +b);
    }

    void val_method()
    {
        c=a+b;
        System.out.println("sum is: "+c );

    }
}
class child extends Parent
{
    int a=200,b=20;
    child(){
            System.out.println("Child second"+a +b);
    }
    void val_method()
    {
        c=a+b;
        System.out.println("sum is: "+c );
    }

}
```

## PROBLEM DESCRIPTION: NEED OF SUPER KEYWORD



```
class Main
{
public static void main (String[] args) {

child obj;
obj=new child();
obj.val_method();
}
}

class Parent
{
    int a=10,b=100,c;
    Parent()
    {
     System.out.println("parent first"+ a +b);
    }

    void val_method()
    {

        c=a+b;
        System.out.println("sum is: "+c);

    }
}
class child extends Parent
{
    int a=200,b=20;
    child(){
             System.out.println("Child second"+a +b);
    }
    void val_method()
    {
        c=super.a+b;
        System.out.println("sum is: "+c );
    }

}
```

**Output**



```
parent first10100
Child second20020
sum is: 30
```

super.a executes parent class variable which is a=10;

## THE SUPER KEYWORD

- It is used to **differentiate the members** of superclass (immediate parent) from the members of subclass, if they have same names.

- **super** can be used to refer immediate parent class instance variable.
- **super** can be used to invoke immediate parent class method.
- **super()** can be used to invoke immediate parent class constructor.

## 1. SUPER CAN BE USED TO INVOKE PARENT CLASS VARIABLE

- The first form of super acts somewhat like this, except that it always refers to the **immediate** superclass variables of the subclass in which it is used.

- This usage has the following general form:

**super.variable**

## 1. SUPER CAN BE USED TO INVOKE PARENT CLASS VARIABLE

```
class Animal{
                String color="white";
             }
class Dog extends Animal{
                String color="black";
                void printColor(){
                System.out.println(color);//prints color of Dog class
                System.out.println(super.color);//prints color of Animal class
                }
        }
class TestSuper1{     //class with a main method
public static void main(String args[]){
Dog d=new Dog();    //object of sub class
d.printColor();
}}
```

## 1. SUPER CAN BE USED TO INVOKE PARENT CLASS VARIABLE

```
class Animal{
                    String color="white";
                }
class Dog extends Animal{
                    String color="black";
                    void printColor(){
                    System.out.println(color);//prints color of Dog class
                    System.out.println(super.color);//prints color of Animal class
                    }
            }
class TestSuper1{     //class with a main method
public static void main(String args[]){
Dog d=new Dog();    //object of sub class
d.printColor();
}}
```

## 1. SUPER CAN BE USED TO INVOKE PARENT CLASS VARIABLE

```
class Animal{
                    String color="white";
                }
class Dog extends Animal{
                    String color="black";
                    void printColor(){
                    System.out.println(color);//prints color of Dog class
                    System.out.println(super.color);//prints color of Animal class
                    }
          }
class TestSuper1{     //class with a main method
public static void main(String args[]){
Dog d=new Dog();    //object of sub class
d.printColor();
}}
```

**Output:**

```
black
white
```

## 2. SUPER CAN BE USED TO INVOKE PARENT CLASS METHOD

- The second form of super acts somewhat like this, except that it always refers to the immediate superclass method of the subclass in which it is used.
- This usage has the following general form:

**super.method()**

## 2. SUPER CAN BE USED TO INVOKE PARENT CLASS METHOD

```
class Animal{
        void eat()
                {System.out.println("eating...");}
        }
class Dog extends Animal{
        void eat()
                {System.out.println("eating bread...");}
        void bark()
                {System.out.println("barking...");}
        void work(){
                super.eat();
                bark();
        }
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

## 2. SUPER CAN BE USED TO INVOKE PARENT CLASS METHOD

```
class Animal{
        void eat()
                {System.out.println("eating...");}
        }
class Dog extends Animal{
        void eat()
                {System.out.println("eating bread...");}
        void bark()
                {System.out.println("barking...");}
        void work(){
                super.eat();
                bark();
        }
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

**Output**

```
eating...
barking...
```

## 3. USING SUPER TO CALL SUPERCLASS CONSTRUCTORS

- A subclass can call a constructor defined by its superclass by use of the following form of super:

  **super(arg-list);**

- Here , arg-list specifies any arguments needed by the constructor in the **superclass.super( )**
- It always be the first statement executed inside a subclass constructor.
- When a subclass calls **super()**, it is calling the constructor of its immediate superclass. Thus , **super()** always refers to the superclass immediately above the calling class.

## 3. USING SUPER TO CALL SUPERCLASS CONSTRUCTORS

```
class Animal{
        Animal(){System.out.println("animal is created");}
        }
class Dog extends Animal{
        Dog(){
        super();  //first statement
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

## 3. USING SUPER TO CALL SUPERCLASS CONSTRUCTORS

```
class Animal{
        Animal(){System.out.println("animal is created");}
        }
class Dog extends Animal{
        Dog(){
        super();  //first statement
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

**Output**

```
animal is created
 dog is created
```

## NOTE:

- Call to super() must be first statement in Derived(Student) Class constructor.
- If a constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass. If the superclass does not have a no-argument constructor, you will get a compile-time error. Object does have such a constructor, so if Object is the only superclass, there is no problem.
- If a subclass constructor invokes a constructor of its superclass, either explicitly or implicitly, you might think that a whole chain of constructors called, all the way back to the constructor of Object. This, in fact, is the case. It is called constructor chaining..

## TYPES OF INHERITANCE IN JAVA

**There are four types of inheritance in java:**

1) **Single**
2) **Multilevel and**
3) **Hierarchical.**
4) **Hybrid**

**In java programming, multiple and hybrid inheritance is supported through interface only.**

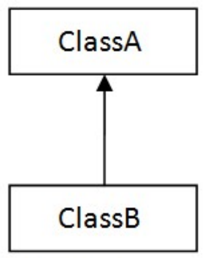
## TYPES OF INHERITANCE IN JAVA

**There are four types of inheritance in java:**

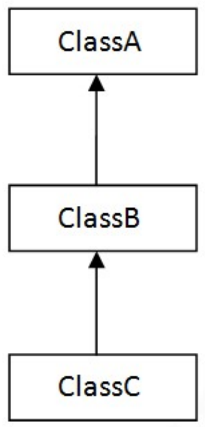
1) **Single**
2) **Multilevel and**
3) **Hierarchical.**
4) **Hybrid**

**In java programming, multiple and hybrid inheritance is supported through interface only.**

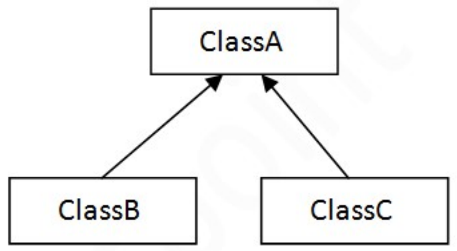
## TYPES OF INHERITANCE IN JAVA



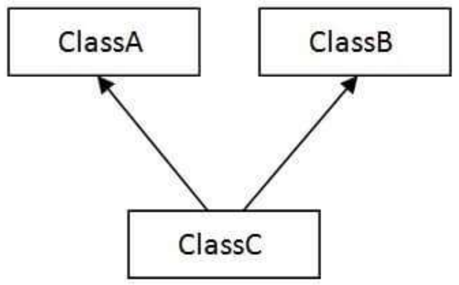
1) Single



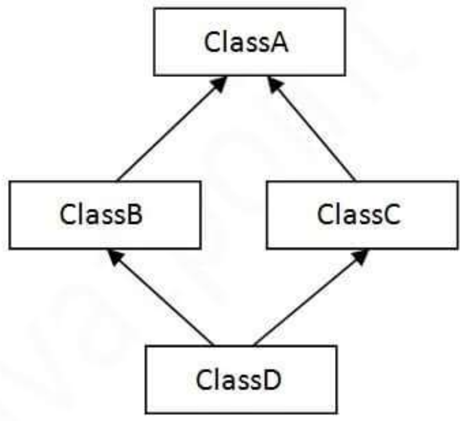
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

## SINGLE INHERITANCE

```
class Animal
{
void eat()
{System.out.println("eating...");}
}

class Dog extends Animal{
void bark()
{System.out.println("barking...");}
}

class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

**Note: When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.**

## MULTILEVEL INHERITANCE

**When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.**

```
class Animal{
        void eat()
                {System.out.println("eating...");}
                }
class Dog extends Animal{
        void bark()
                {System.out.println("barking...");}
                }
class BabyDog extends Dog{
        void weep(){System.out.println("weeping...");}
                }
class TestInheritance2{
public static void main(String args[]){
        BabyDog d=new BabyDog();
                d.weep();
                d.bark();
                d.eat();
                }
                }
```

## MULTILEVEL INHERITANCE

```
class Animal{
        void eat()
                {System.out.println("eating...");}
                }
class Dog extends Animal{
        void bark()
                {System.out.println("barking...");}
                }
class BabyDog extends Dog{
        void weep(){System.out.println("weeping...");}
                }
class TestInheritance2{
public static void main(String args[]){
        BabyDog d=new BabyDog();
                d.weep();
                d.bark();
                d.eat();
                }
                }
```

**When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.**

**OUTPUT:**

```
weeping...
barking...
eating...
```

## HIERARCHICAL INHERITANCE

**When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.**

```
class Animal{
            void eat()
                        {System.out.println("eating...");}
                        }
class Dog extends Animal{
            void bark()
                        {System.out.println("barking...");}
                        }
class Cat extends Animal{
            void meow()
                        {System.out.println("meowing...");}
                        }
class TestInheritance3{
public static void main(String args[]){
            Cat c=new Cat();
            c.meow();
            c.eat();
            //c.bark();//C.T.Error
            }}
```

## HIERARCHICAL INHERITANCE

```
class Animal{
        void eat()
                {System.out.println("eating...");}
                }
class Dog extends Animal{
        void bark()
                {System.out.println("barking...");}
                }
class Cat extends Animal{
        void meow()
                {System.out.println("meowing...");}
                }
class TestInheritance3{
public static void main(String args[]){
        Cat c=new Cat();
        c.meow();
        c.eat();
        //c.bark();//C.T.Error
        }}
```

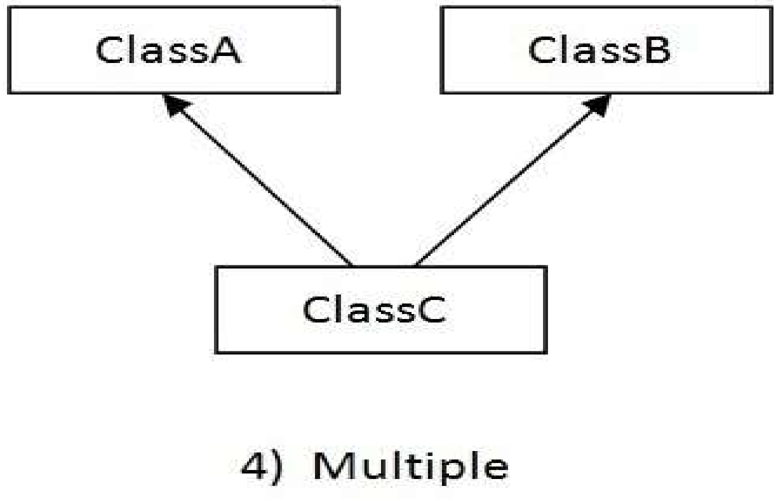
**When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.**

**OUTPUT**

```
meowing...
eating...
```

## MULTIPLE INHERITANCE

When a class extends multiple classes i.e., known as multiple inheritance. For Example:



It is not allowed in Java through extends

.

## WHY MULTIPLE INHERITANCE IS NOT SUPPORTED IN JAVA?

- Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object.
- There will be ambiguity to call method of A or B class.

## WHY MULTIPLE INHERITANCE IS NOT SUPPORTED IN JAVA?

```
class A{
          void add()
                    {a =10;
                    b=5;
                    c=a+b;}                  }
class B{
          void add()
                    {a =10;
                    b=5;
                    c=15;
                    d=a+b+c;}
class C extends A,B{//suppose if it were
    Public Static void main(String args[]){
                     C obj=new C();
                    obj.add();//Now which add() method would be invoked?
                    }         }
```

## WHY MULTIPLE INHERITANCE IS NOT SUPPORTED IN JAVA?

- To reduce the complexity and
- simplify the language,

  **multiple inheritance is not supported in java.**
- Since compile time errors are better than runtime errors,
- java renders compile time error if you inherit 2 classes.

- So whether you have same method or different, there will be compile time error now.
- Therefore, Inheritance is called Compile Time Mechanism.

```
class A{
        void add()
                {a =10;
                b=5;
                c=a+b;}
                }
class B{
        void sub()
                {
                a =10;
                b=5;
                c= a-b;
        }
class C extends A,B{
    Public Static void main(String args[]){
                 C obj=new C();
                obj.add();
                }
        }
```

**Output:**
**'{' expected**
**class C extends A,B**

THANK YOU

?