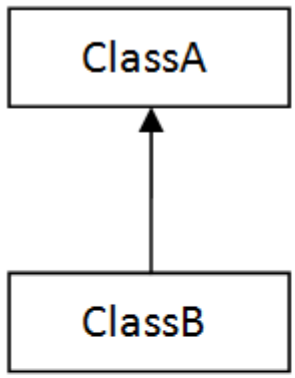
Unit 4

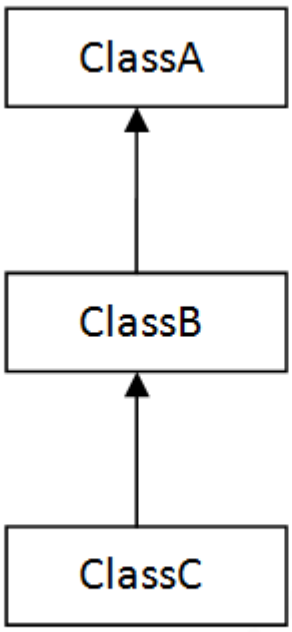
Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

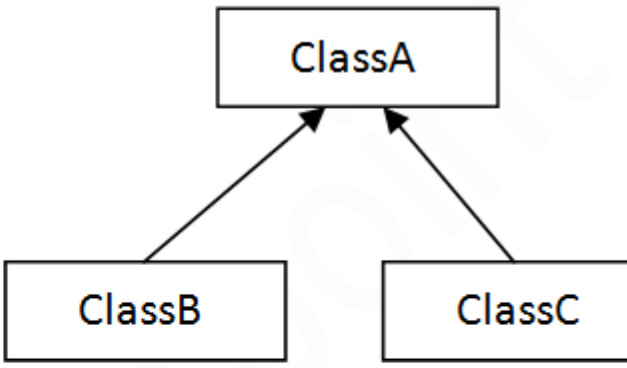
In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



1) Single



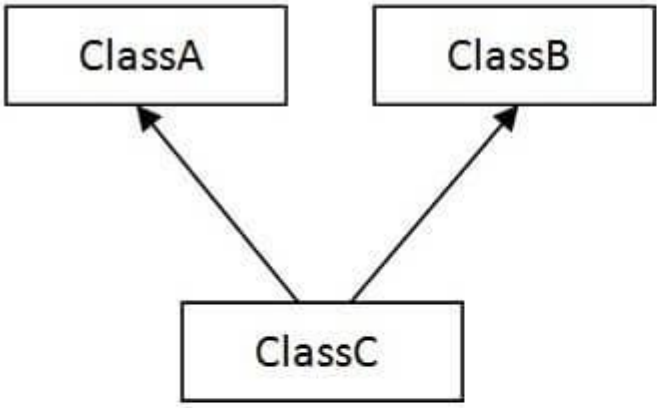
2) Multilevel



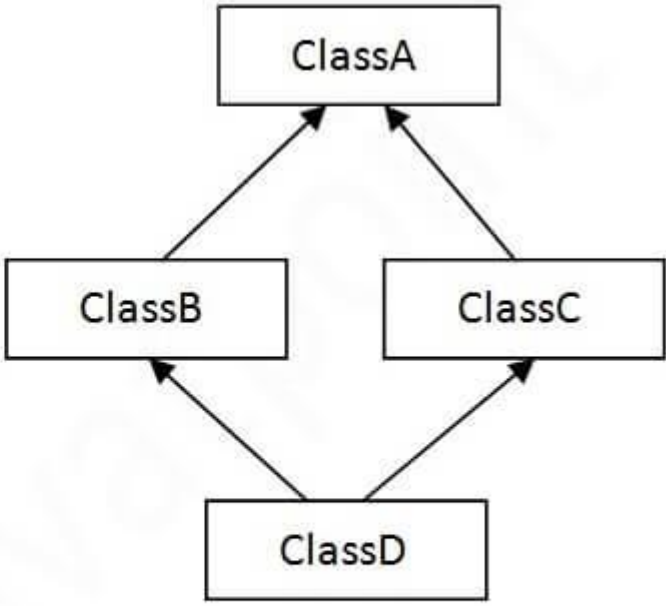
3) Hierarchical

> Note: Multiple inheritance is not supported in Java through class.

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

.

Single Inheritance Example

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.

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class TestInheritance{
8. public static void main(String args[]){
9. Dog d=new Dog();
10.d.bark();
11.d.eat();
```

```
12.}}
```

Output:

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

Multilevel Inheritance Example

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.

```
1. class Animal
2. {
3. void eat()
4. {
5. System.out.println("eating...");}
6. }
7. class Dog extends Animal
8. {
9. void bark()
10.{
11.System.out.println("barking...");}
12.}
13.class BabyDog extends Dog{
14.void weep()
15.{
16.System.out.println("weeping...");
17.}
18.}
19.class TestInheritance2
20.{
21.public static void main(String args[])
```

```
{
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}
}
```

Output:

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

Hierarchical Inheritance Example

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()
{
```

```
16. System.out.println("meowing...");}
17. }
18. class TestInheritance3
19. {
20. public static void main(String args[])
21. {
22. Cat c=new Cat();
23. c.meow();
24. c.eat();
25. //c.bark();//C.T.Error
26. }
27. }
```

Output:

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

Hybrid Inheritance Example

Hybrid Inheritance (Through Interfaces): **It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritances with classes, hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.**

Hybrid Inheritance implement two types of inheritance(single and hierarchical) together to form hybrid inheritance.
Class A and B extends class C → Hierarchical inheritance
Class D extends class A → Single inheritance

```
class C
{
   public void disp()
   {
        System.out.println("C");
   }
}
```

```
class A extends C
{
   public void disp()
   {
         System.out.println("A");
   }
}

class B extends C
{
   public void disp()
   {
         System.out.println("B");
   }

}

class D extends A
{
   public void disp()
   {
         System.out.println("D");
   }
   public static void main(String args[]){

         D obj = new D();
         obj.disp();
   }
}
```

Output:

```
D
```

Order of invocation

Whenever we create an object of a class, the default constructor of that class is invoked automatically to initialize the members of the class.
If we inherit a class from another class and create an object of the derived class, it is clear that the default constructor of the derived class will be invoked but before that the default constructor of all of the base classes will be invoke, i.e. the order of invocation is that the base class's default constructor will be invoked first and then the derived class's default constructor will be invoked.

A client request invocation**. An** invocation **is a request that has been prepared and is ready for execution.** Invocations **provide a generic**

(command) interface that enables a separation of concerns between the creator and the submitter.

Base class constructors are called first and the derived class constructors are called next in single inheritance.

Protected data members

Protected data member **and method are only accessible by the classes of the same package and the subclasses present in any package. You can also say that the** protected **access modifier is similar to default access modifier with one exception that it has visibility in sub classes.**

Private data members

The private **access modifier is specified using the keyword** private**. The methods or** data members **declared as** private **are accessible only within the class in which they are declared. Any other class of the same package will not be able to access these** members

Public data members

`Public` is a Java keyword which declares a member's access as public. Public members are visible to all other classes. This means that any other class can access a `public` field or method. Further, other classes can modify `public` fields unless the field is declared as `final`.