SNS COLLEGE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTION)
COIMBATORE – 35
DEPARTMENT OF COMPUTER SIENCE AND ENGINEERING

UNIT III

INHERITANCE AND POLYMORPHISM

Inheritance in Java

1. Inheritance
2. Types of Inheritance
3. Why multiple inheritance is not possible in Java in case of class?

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

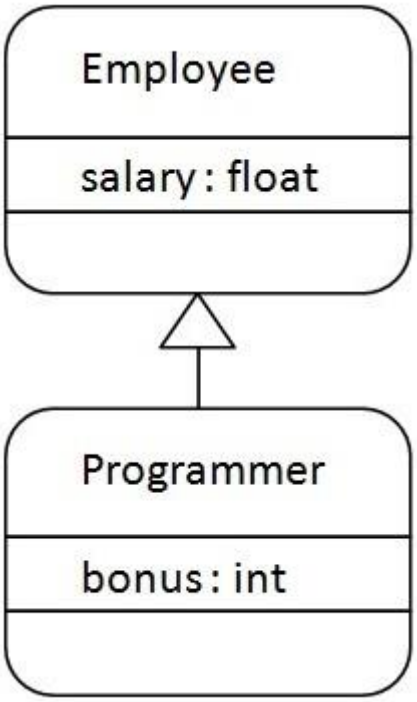
```
class Subclass-name extends Superclass-name
{
  //methods and fields
```

4. }

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Java Inheritance Example



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

```
1.  class Employee
2.  {
3.   float salary=40000;
4.  }
5.
6.  class Programmer extends Employee
7.  {
8.   int bonus=10000;
9.   public static void main(String args[])
10. {
11.    Programmer p=new Programmer();
12.    System.out.println("Programmer salary is:"+p.salary);
13.    System.out.println("Bonus of Programmer is:"+p.bonus);
```

14. }
15. }

Test it Now

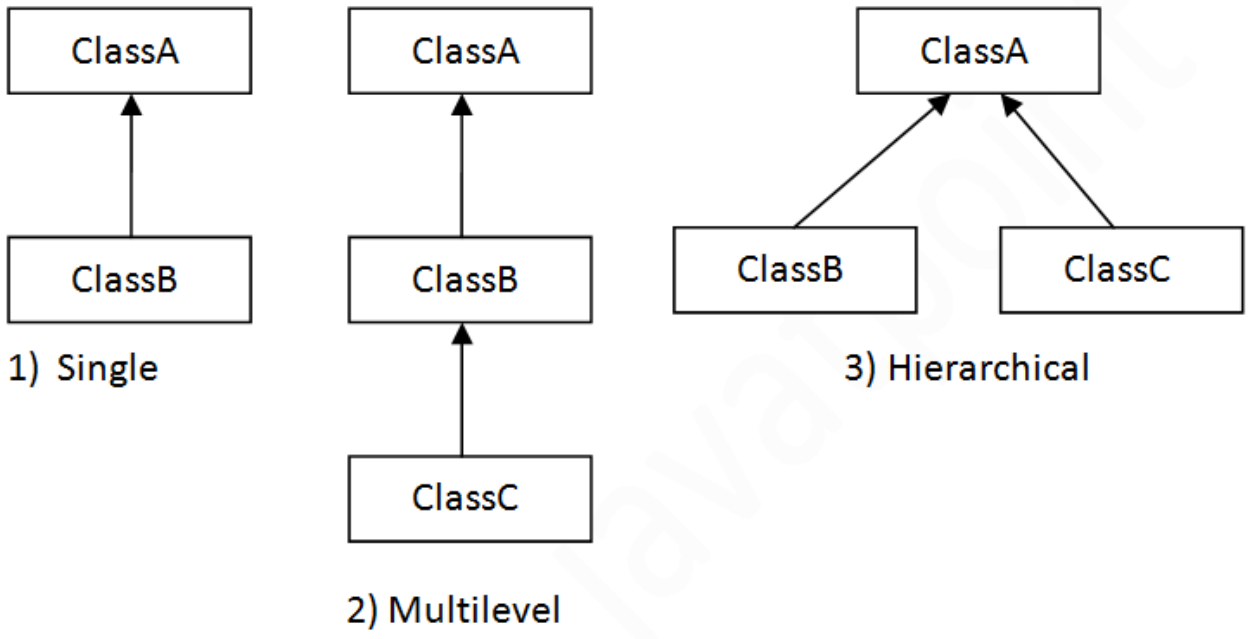
```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

Types of inheritance in java

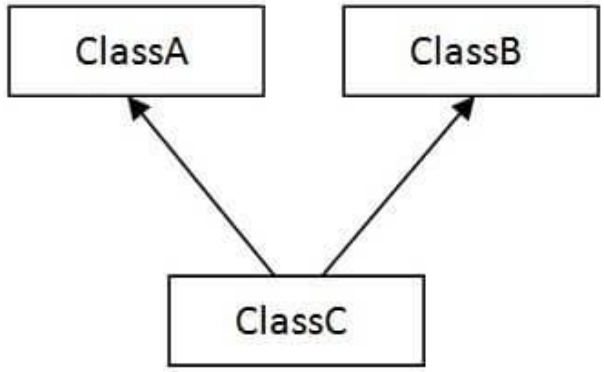
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

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

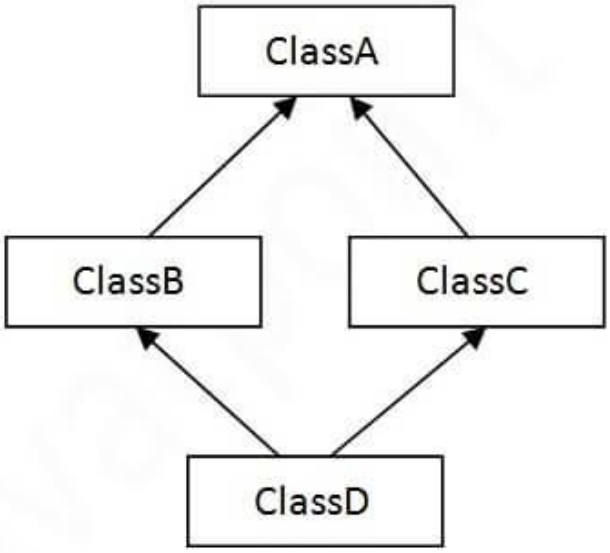


Note: Multiple inheritance is not supported in Java through class.

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.

File: TestInheritance.java

```
1.  class Animal
2.  {
3.  void eat()
4.      {
5.      System.out.println("eating...");
6.      }
7.  }
8.
9.  class Dog extends Animal
10. {
11. void bark()
12.     {
13.     System.out.println("barking...");
14.     }
15. }
```

```
16. class TestInheritance
17. {
```

```
18. public static void main(String args[])
19.    {
20.     Dog d=new Dog();
21.    d.bark();
22.     d.eat();
23.     }
24. }
```

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.

File: TestInheritance2.java

```
1.  class Animal
2.  {
3.  void eat()
4.  {
5.  System.out.println("eating...");
6.  }
7.  }
8.
9.  class Dog extends Animal
10. {
11. void bark()
12. {
13. System.out.println("barking...");
14. }
15. }
16.
17. class BabyDog extends Dog
18. {
19. void weep()
20. {
21. System.out.println("weeping...");
22. }
23. }
24. class TestInheritance2
25. {
26. public static void main(String args[])
27. {
28. BabyDog d=new BabyDog();
29. d.weep();
```

```
30. d.bark();
31. d.eat();
32. }
33. }
```

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.

File: TestInheritance3.java

```
1.  class Animal
2.  {
3.  void eat()
4.  {
5.  System.out.println("eating...");
6.  }
7.  }
8.
9.  class Dog extends Animal
10. {
11. void bark()
12. {
13. System.out.println("barking...");
14. }
15. }
16.
17. class Cat extends Animal
18. {
19. void meow()
20. {System.out.println("meowing...");}
21. }
22. class TestInheritance3
23. {
24. public static void main(String args[])
25. {
26. Cat c=new Cat();
27. c.meow();
28. c.eat();
29. //c.bark();//C.T.Error
30. }
31. }
```

Output:

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

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, 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 the same method and you call it from child class object, there will be ambiguity to call the method of A or B class. 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.

```
1. class A
2. {
3. void msg(){System.out.println("Hello");}
4. }
5. class B
6. {
7. void msg(){System.out.println("Welcome");}
8. }
9.
10. class C extends A,B{//suppose if it were
11.     public static void main(String args[])
12. {
13.    C obj=new C();
14.    obj.msg();//Now which msg() method would be invoked?
15. }
16. }
```

Test it Now

```
Compile Time Error
```