Java Enterprise Edition

ANUDIP FOUNDATION

INHERITANCE - Is -A

Objective: • Inheritance concepts • Types • Aggregation	**Materials Required:** 1. Eclipse IDE/IntelliJ/STC
Theory:60mins	**Practical:30mins**
Total Duration: 90mins	

INHERITANCE - Is -A

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.

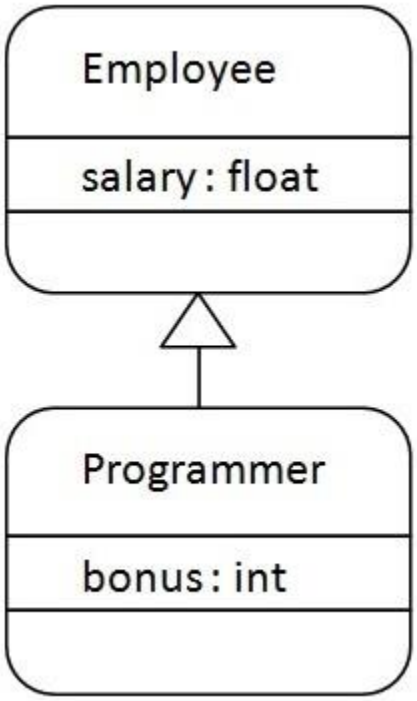
The syntax of Java Inheritance

```
1.  class Subclass-name extends Superclass-name
2.  {
3.    //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.

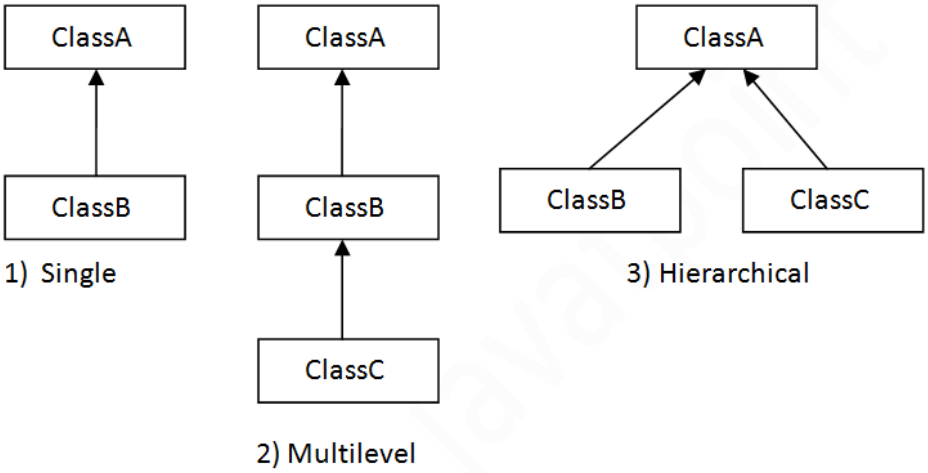
```
1.  class Employee{
2.   float salary=40000;
3.  }
4.  class Programmer extends Employee{
5.   int bonus=10000;
6.   public static void main(String args[]){
7.     Programmer p=new Programmer();
8.     System.out.println("Programmer salary is:"+p.salary);
9.     System.out.println("Bonus of Programmer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

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

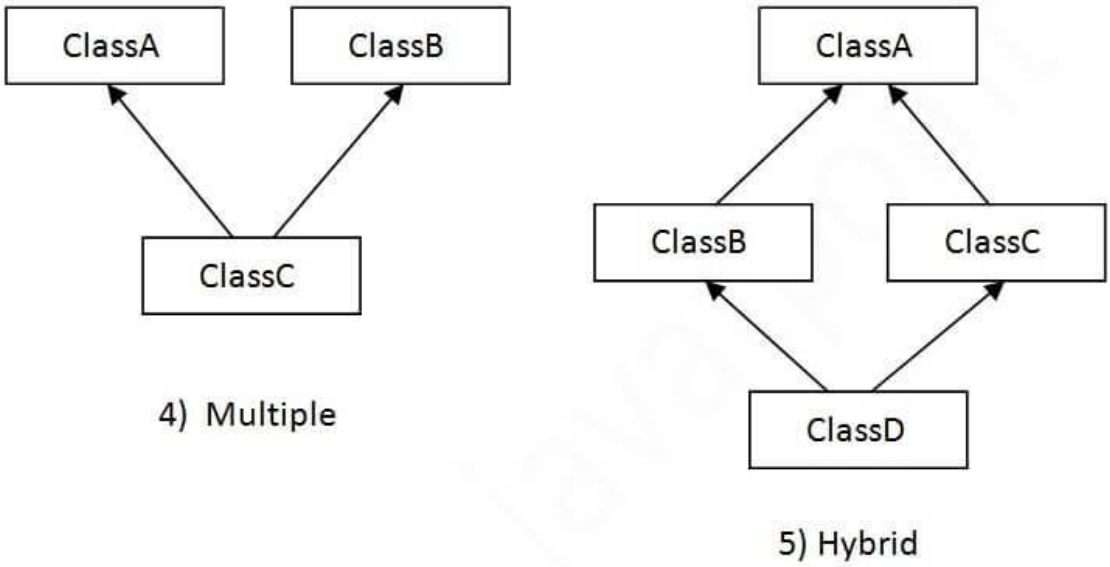
Types of inheritance in java

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.



.

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.  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.

File: TestInheritance2.java

```
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 BabyDog extends Dog{
8.      void weep(){System.out.println("weeping...");}
9.      }
10.     class TestInheritance2{
11.     public static void main(String args[]){
12.     BabyDog d=new BabyDog();
13.     d.weep();
14.     d.bark();
15.     d.eat();
16.     }}
```

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.      void eat(){System.out.println("eating...");}
3.      }
4.      class Dog extends Animal{
5.      void bark(){System.out.println("barking...");}
6.      }
7.      class Cat extends Animal{
8.      void meow(){System.out.println("meowing...");}
9.      }
10.     class TestInheritance3{
11.     public static void main(String args[]){
```

```
12.     Cat c=new Cat();
13.     c.meow();
14.     c.eat();
15.     //c.bark();//C.T.Error
16.     }}
```

Output:

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

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

Compile Time Error

INHERITANCE – Has – A

Aggregation in Java

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

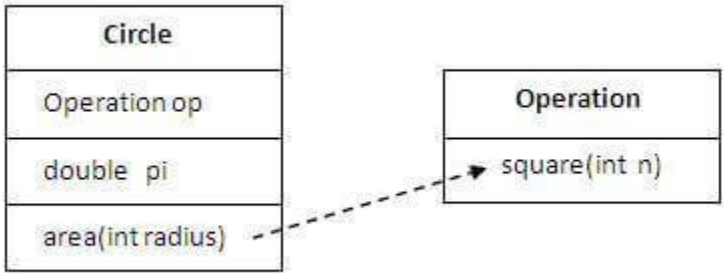
```
1.  class Employee{
2.  int id;
3.  String name;
4.  Address address;//Address is a class
5.  ...
6.  }
```

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

Why use Aggregation?

- For Code Reusability.

Simple Example of Aggregation



In this example, we have created the reference of Operation class in the Circle class.

```
1.  class Operation{
2.   int square(int n){
3.    return n*n;
4.   }
```

```
5.      }
6.
7.      class Circle{
8.       Operation op;//aggregation
9.       double pi=3.14;
10.
11.      double area(int radius){
12.        op=new Operation();
13.        int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).
14.        return pi*rsquare;
15.      }
16.
17.
18.
19.      public static void main(String args[]){
20.        Circle c=new Circle();
21.        double result=c.area(5);
22.        System.out.println(result);
23.      }
24.     }
```

Output :

Output:78.5

When use Aggregation?

- Code reuse is also best achieved by aggregation when there is no is-a relationship.
- Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Understanding meaningful example of Aggregation

In this example, Employee has an object of Address, address object contains its own informations such as city, state, country etc. In such case relationship is Employee HAS-A address.

Address.java

```
1.      public class Address {
2.      String city,state,country;
3.
4.      public Address(String city, String state, String country) {
5.         this.city = city;
6.         this.state = state;
7.         this.country = country;
8.      }
9.
10.     }
```

Emp.java

```
1.      public class Emp {
2.      int id;
3.      String name;
4.      Address address;
5.
6.      public Emp(int id, String name,Address address) {
7.         this.id = id;
8.         this.name = name;
9.         this.address=address;
10.     }
11.
12.     void display(){
13.     System.out.println(id+" "+name);
14.     System.out.println(address.city+" "+address.state+" "+address.country);
15.     }
16.
17.     public static void main(String[] args) {
18.     Address address1=new Address("gzb","UP","india");
19.     Address address2=new Address("gno","UP","india");
20.
21.     Emp e=new Emp(111,"varun",address1);
```

```
22.       Emp e2=new Emp(112,"arun",address2);
23.
24.       e.display();
25.       e2.display();
26.
27.       }
28.       }
```

Output:

```
111 varun
gzb UP india
112 arun
gno UP india
```