Object Oriented Programming

Inheritance, Association & Composition

Compiled By: Umair Yaqub

Inheritance in Java:

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

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 parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

Why use inheritance in java

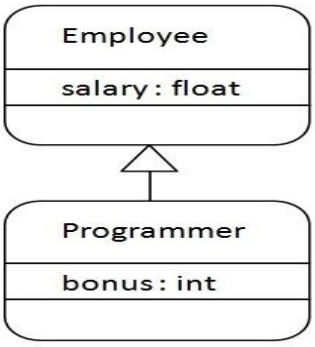
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Syntax of Java Inheritance

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

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 parent or super class 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. Relationship between two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

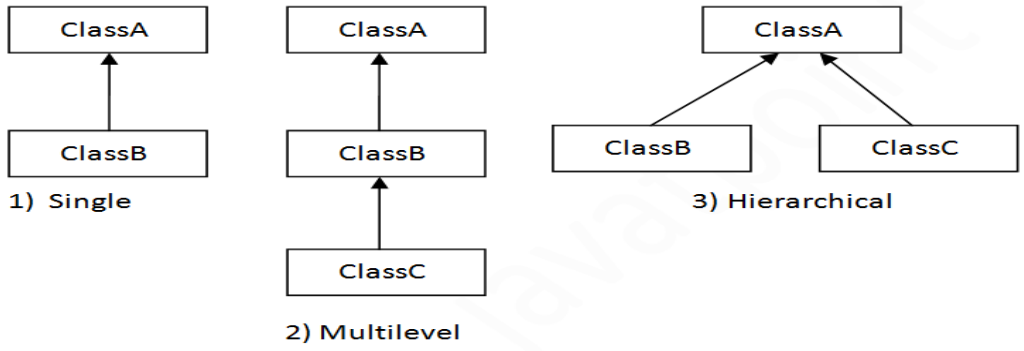
```
class Employee{
        float salary=40000;
        }
class Programmer extends Employee{
        int bonus=10000;
        public static void main(String args[]){
                Programmer p=new Programmer();
                System.out.println("Programmer salary is:"+p.salary);
                System.out.println("Bonus of Programmer is:"+p.bonus);
                }
        }
```

```
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.



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

Single Inheritance Example:

```
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();
        }
    }
```

Output: barking...

eating...

Multilevel Inheritance Example:

```
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();
    }
}
```

Output:

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

Hierarchical Inheritance Example:

```
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
        }
}
```

Output: meowing...

eating...

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritances is not supported in java.

Aggregation in Java:

If a class has 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.

```
class Employee{
        int id;
        String name;
        Address address;//Address is a class
```

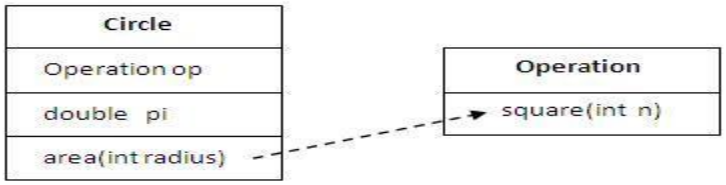
```
    ...
}
```

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




```
class Operation{
    int square(int n){
    return n*n;
    }
}

class Circle{
    Operation op;//aggregation
    double pi=3.14;
    double area(int radius){
        op=new Operation();
        int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).
        return pi*rsquare;
 }
    public static void main(String args[]){
        Circle c=new Circle();
        double result=c.area(5);
        System.out.println(result);
     }
}
```

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.

```
public class Address {
        String city,state,country;
        public Address(String city, String state, String country) {
                this.city = city;
                this.state = state;
                this.country = country;
        }
}
public class Emp {
        int id;
        String name;
        Address address;
        public Emp(int id, String name,Address address) {
                this.id = id;
                this.name = name;
                this.address=address;
        }
        void display(){
                System.out.println(id+" "+name);
                System.out.println(address.city+" "+address.state+" "+address.country);
        }
public static void main(String[] args) {
        Address address1=new Address("gzb","UP","india");
        Address address2=new Address("gno","UP","india");
        Emp e=new Emp(111,"varun",address1);
        Emp e2=new Emp(112,"arun",address2);
        e.display();
        e2.display();
        }
}
```

Output:

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