CHAPTER

# 15

# INHERITANCE

It is possible to acquire all the members of a class and use them in another class by relating the objects of the two classes. This is possible by using inheritance concept. When a class is written by a programmer and another programmer wants the same features (members) in his class also, then the other programmer will go for inheritance. This is done by deriving the new class from the existing class. This chapter deals with how to derive new classes from existing classes and the advantage of inheritance.

**Important Interview Question**

*What is inheritance?*

*Deriving new classes from existing classes such that the new classes acquire all the features of existing classes is called inheritance.*

## Inheritance

Inheritance is a concept where new classes can be produced from existing classes. The newly created class acquires all the features of existing class from where it is derived.

Let us take an example to understand the inheritance concept. A programmer in a software development team has written the `Teacher` class as shown here:

```
//Teacher class
class Teacher
{
      //instance variables
      int id;
      String name;
      String address;
      float sal;

      //setter method to store id
      void setId(int id)
      {
            this.id=id;
      }

      //getter method to retrieve id
      int getId()
      {
            return id;
```

```
        }

        //to store name
        void setName(String name)
        {
                this.name=name;
        }

        //to retrieve name
        String getName()
        {
                return name;
        }

        //to store address
        void setAddress(String address)
        {
                this.address=address;
        }

        //to retrieve address
        String getAddress()
        {
                return address;
        }

        //to store salary
        void setSal(float sal)
        {
                this.sal= sal;
        }

        //to retrieve salary
        float getSal()
        {
                return sal;
        }
}
```

In the preceding code, observe that all the `setter()` methods are mutator methods, and t `getter()` methods are accessor methods. Save the preceding code as `Teacher.java` and compile to get `Teacher.class`. These files are then copied into a central database that is available to eve programmer in the project development team. So other programmers also know th `Teacher.class` is available in the database. This situation is shown in Figure 15.1.

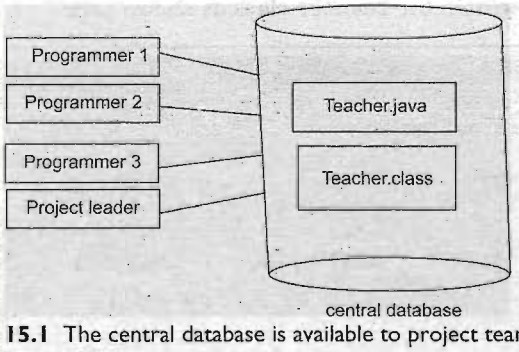


**Figure 15.1** The central database is available to project team members

Now, another programmer wants to develop a `Student` class as part of the project. The programı is not considering the `Teacher` class which is already available in the central database, and writ the Student class as shown here:

```
//Student class - version 1
class Student
{
     //instance variables
     int id;
     String name;
     String address;
     int marks;

//setter method to store id
void setId(int id)
{
     this.id=id;
}

//getter method to retrieve id
int getId()
{
     return id;
}

//to store name
void setName(String name)
{
     this.name=name;
}

//to retrieve name
String getName()
{
     return name;
}

//to store address
void setAddress(String address)
{
     this.address=address;
}

//to retrieve address
String getAddress()
{
     return address;
}

//to store marks
void setMarks(int marks)
{
     this.marks=marks;
}

//to retrieve marks
int getMarks()
{
     return marks;
}
}
```

Save the preceding code as `Student.java` and compile it to get `Student.class`. Now, if the programmer wants to use `Student.class`, he can write another program where he can create an object to Student and use the features available, as shown in the next program.

**Program 1:** Write a program to create an object to Student class, then store data into it and retrieve and display the data.

*Note*

*Already created Student class is used in this program.*

```
//Using Student class
class Use
{
      public static void main(String args[ ])
      {
            //create an object to Student class
            Student s = new Student();

            //store data into object- for this use setter methods
            s.setId(1001);
            s.setName("Chandra Sekhar");
            s.setAddress("MIG-12, Kukatpally, Hyderabad");
            s.setMarks(950);

            //retrieve data using getter methods and display
            System.out.println("Id= "+s.getId());
            System.out.println("Name= "+s.getName());
            System.out.println("Address= "+s.getAddress());
            System.out.println("Marks= "+s.getMarks());
      }
}
```

Output:

```
C:\> javac Use.java
C:\> java Use
Id= 1001
Name= Chandra Sekhar
Address= MIG-12, Kukatpally, Hyderabad
Marks= 950
```

In this program, an object to Student class is created by JVM, as shown Figure 15.2.

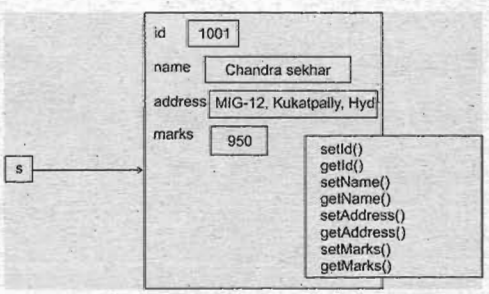


**Figure 15.2** Student class object

Just compare the Teacher class and Student class. You can find 75% similarities in classes. While developing the Student class, if the programmer has thought of reusing Teac code, developing Student class would have been very easy. With this idea, let us rewrite class again. Whatever code is available in Teacher class will be omitted in writing the Stud as that code will be automatically available to Student class. For this purpose, simply keyword 'extends' as:

```
class Student extends Teacher
```

The preceding statement means all the members of `Teacher` class are available to `Student` class without rewriting them in `Student` class. Only additional members should be written in `Student` class. So, developing the `Student` class will become easy as shown in Program 2.

**Program 2:** Write a program to use 'extends' keyword to create Student class by reusing Teacher class code. We should write only additional members in Student class which are not available in Teacher class.

```
//Student class - version 2
class Student extends Teacher
{
	//Since id, name, address are available from Teacher class, we omit
	//those instance variables and the corresponding methods.
	int marks;

	//to store marks
	void setMarks(int marks)
	{
		this.marks=marks;
	}

	//to retrieve marks
	int getMarks()
	{
		return marks;
	}
}
```

Save the preceding code as `Student.java` and compile it to get `Student.class`. In other words, we can say `Student` class is created based on `Teacher` class.

Like this, creating new classes from existing classes is called inheritance. The original class, i.e., `Teacher` class is called `super` class and the newly created class, i.e., `Student` class is called subclass. We created `Student` class by extending the `Teacher` class, as:

```
class Student extends Teacher
```

So, the syntax (correct format) of inheritance is:

```
class subclass extends superclass
```

Now, please go through `Use.java` again. Execute it, and you will see the same result. In `Use.java`, we create `Student` class object, as:

```
Student s = new Student();
```

When an object to `Student` class is created, it contains a copy of `Teacher` class within it. This means there is a relation between the `Teacher` class and `Student` class objects. This is the reason why Teacher class members are available to Student class. Note that we do not create Teacher class object, but still a copy of it is available to Student class object. Please see the object diagram of Student class in Figure 15.3. You can understand that all the members (i.e., variables and methods) of Teacher class as well as Student class are available in it.

**Important Interview Question**

*Why super class members are available to sub class?*

*Because, the sub class object contains a copy of super class object.*

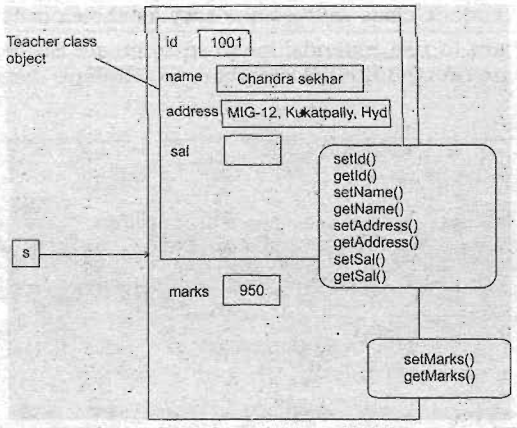


**Figure 15.3** Student class object contains a copy of Teacher class

Then, what is the advantage of inheritance? Please look at `Student` class version 1 and `Stude` class version 2. Clearly, second version is smaller and easier to develop. By using inheritance, programmer can develop the classes very easily. Hence programmer's productivity is increased. I can deliver more code in less time. This will increase the overall productivity of the Organizatic and hence more growth for the Organization.

**Important Interview Question**

*What is the advantage of inheritance?*

*In inheritance, a programmer reuses the super class code without rewriting it, in creation of sub classes. So, developing the classes becomes very easy. Hence, the programmer's productivity is increased.*

## The Keyword 'super'

If we create an object to super class, we can access only the super class members, but not the s class members. But if we create sub class object, all the members of both super and sub classes available to it. This is the reason, we always create an object to sub class in inheritance. So times, the super class members and sub class members may have same names. In that case, default only sub class members are accessible. This is shown in the following example program.

**Program 3:** Write a program where the names of instance variables and methods in super and s classes are same. Hence, by default only sub class members are accessible.

```
//By default sub class members are accessible
//to subclass object
class One
{
     //super class var
     int i=10;

     //super class method
     void show()
```

```
        {
                System.out.println("super class method:i= "+i);
        }
}
class Two extends One
{
        //sub class var
        int i=20;

        //sub class method
        void show()
        {
                System.out.println("sub class method:i= "+i);
        }
}

class Super1
{
        public static void main(String args[ ])
        {
                //create sub class object
                Two t = new Two();

                //This will call sub class method only
                t.show();
        }
}
```

Output:

```
C:\> javac Super1.java
C:\> java Super1
sub class method:i= 20
```

Please observe that a call to sub class method, `t.show();` calls and executes only sub class method. And hence the sub class instance variable i value 20 is displayed. In such a case, how to access the super class members from sub class is the question. For this purpose, `super` key word has been invented. `super` refers to super class members from a sub class. For example,

- `super` can be used to refer to super class variables, as:

```
super.variable
```

- `super` can be used to refer to super class methods, as:

```
super.method()
```

- `super` can be used to refer to super class constructor.

We need not call the default constructor of the super class, as it is by default available to sub class. To call the parameterized constructor, we can write:

```
super(values);
```

In the next program, we want to access super class instance variable and super class method directly in sub class using `super` key word.

**Program 4:** Write a program to access the super class method and instance variable by using super key word from sub class.

```
//super - to access the super class method and variable
```

```
class One
{
	//super class var
	int i=10;

	//super class method
	void show()
	{
		System.out.println("super class method:i= "+i);
	}
}
class Two extends One
{
	//sub class var
	int i=20;

	//sub class method
	void show()
	{
		System.out.println("sub class method:i= "+i);

		//using super to call super class method
		super.show();

		//using super to access super class var
		System.out.println("super i= "+ super.i);
	}
}

class Super1
{
	public static void main(String args[ ])
	{
		//create sub class object
		Two t = new Two();

		//This will call sub class method only
		t.show();
	}
}
```

Output:

```
C:\> javac Super1.java
C:\> java Super1
sub class method:i= 20
super class method:i= 10
super i= 10
```

Now the next thing is to access the constructors of the super class. We need not access the constructor of the super class, as it is available to sub class by default. This is illustrated following program.

**Program 5:** Write a program to prove that the default constructor of the super class is avail sub class by default.

```
//Calling super class default constructor from sub class.
class One
{
	//super class default constructor
	One()
	{
		System.out.println("One");
	}
}
class Two extends One
```

```
{
        //sub class default constructor
        Two()
        {
                System.out.println("Two");
        }
}

class Super1
{
        public static void main(String args[])
        {
                //create sub class object
                Two t = new Two();
        }
}
```

Output:

```
C:\> javac Super1.java
C:\> java Super1
One
Two
```

Please observe that when sub class object is created, first of all the super class default constructor is called and then only the sub class constructor is called.

In the following program, we take a parameterized constructor in the super class. This is not available to sub class by default. So it should be called by using `super` keyword.

**Program 6:** Write a program to understand that the parameterized constructor of the super class can be called from sub class using `super()`.

```
//Calling super class parameterized constructor from sub class.
class One
{
        //super class var
        int i;

        //super class para constructor
        One(int i)
        {
                this.i = i;
        }

}
class Two extends One
{
        //sub class var
        int i;

        //sub class para constructor
        Two(int a,int b)
        {
                super(a); //call super class constructor and pass a.
                i = b; //initialize sub class var
        }

        //sub class method
        void show()
        {
                System.out.println("sub class i= "+ i);
                System.out.println("super class i= "+ super.i);
        }
}
```

```
class Super1
{
      public static void main(String args[ ])
      {
            //create sub class object
            Two t = new Two(11, 22);

            //call sub class method
            t.show();

      }
}
```

Output:

```
C:\> javac Super1.java
C:\> java Super1
sub class i= 22
super class i= 11
```

In the preceding program, there is a parameterized constructor in super class which initializes instance variable i, as:

```
One(int i)
{
      this.i = i;
}
```

This constructor can be called from sub class by writing another parameterized constructor in class and passing two values to it, as:

```
Two(int a, int b)
{
      super(a); //call super class constructor and pass a.
      i = b; //initialize sub class var
}
```

The preceding constructor will receive data when the sub class object is created, as:

```
Two t = new Two(11, 22);
```

So, the values 11 and 22 are copied into a and b respectively. And then, the value 'a' is sent to super class parameterized constructor by using the statement:

```
super(a);
```

One condition is that the statement calling the super class constructor should be the first in sub class constructor. This implies that the super class constructor should be given priority the sub class constructor.

## The Protected Specifier

The `private` members of the super class are not available to sub classes directly. But some there may be a need to access the data of super class in the sub class. For this purpose, `protected` specifier is used. `protected` is commonly used in super class to make the members of the class available directly in its sub classes. We can think that the `protected` specifier works `public` with respect to sub classes. See the program 7.

**Program 7:** Write a program to understand private members are not accessible in sub class, but protected members are available in sub class.

```
//private and protected
class Access
{
      private int a;
      protected int b;
}
class Sub extends Access
{
      public void get()
      {
            System.out.println(a);  //error - a is private
            System.out.println(b);
      }
}

class Test
{
      public static void main(String args[])
      {
            Sub s = new Sub();
            s.get();
      }
}
```

Output:

```
C:\> javac Test.java
Test.java:11: a has private access in Access
      System.out.println(a);
                         ^
1 error
```

In the next program, we create Shape class with a protected instance variable 'l'. From Shape class, we derive Square class and from Square class, we derive Rectangle class. We want to calculate areas of Square and Rectangle classes. For this purpose, in Square class we can directly use 'l' of the Shape class and calculate the area of square as l*l. We can also calculate area of rectangle, by taking another variable 'b' and using the 'l' of Shape, as l*b.

**Program 8:** Write a program to find the areas of Square and Rectangle by deriving them from Shape.

```
//Shape is the super class for Square
//And Square is the super class for Rectangle
class Shape
{
      //take protected type var
      protected double l;

      //parameterized constructor
      Shape(double l)
      {
            this.l = l;
      }
}

class Square extends Shape
{
      //call Shape's constructor and send l value
      Square(double l)
      {
            super(l);
```

```
        }

        //calculate area of square
        void area()
        {
                //because of inheritance, 'l' of Shape class is available
                System.out.println("Area of Square= "+ (l*l));
        }
}

class Rectangle extends Square
{
        //var
        private double b;

        //call Square's constructor and send x value
        Rectangle(double x,double y)
        {
                super(x);
                b = y;
        }

        //calculate area of rectangle
        void area()
        {
                //because of inheritance,  'l' of Shape class is available
                System.out.println("Area of Rectangle= "+ (l*b));
        }
}

class Inherit
{
        public static void main(String args[])
        {
                //display area of square
                Square s = new Square(5.5);
                s.area();

                //display area of rectangle
                Rectangle r = new Rectangle(5.5, 6);
                r.area();
        }
}
```

Output:

```
C:\> javac Inherit.java
C:\> java Inherit
Area of Square= 30.25
Area of Rectangle= 33.0
```

# Types of Inheritance

So far, we discussed about inheritance and the advantage of using inheritance in softw development. Let us now look at the types of inheritance. There are two types of inheritance, Sin and Multiple.

- **Single Inheritance:** Producing sub classes from a single super class is called sin inheritance. In this, a single super class will be there. There can be one or more sub class See the examples in Figure 15.4.

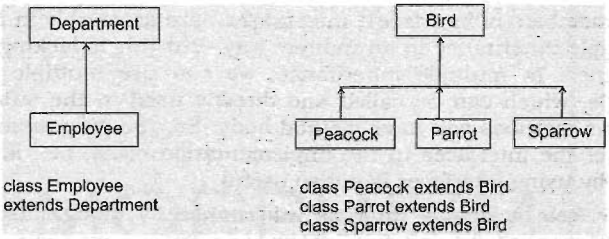


**Figure 15.4** Single inheritance •

- **Multiple Inheritance:** Producing sub classes from multiple super classes is called multiple inheritance. In this case, there will be more than one super class and there can be one or more sub classes. See the examples in Figure 15.5.

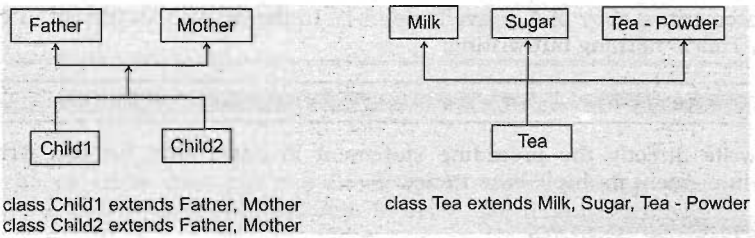


**Figure 15.5** Multiple inheritance.

Only Single inheritance is available in Java. There is no multiple inheritance in Java. The following are the reasons:

- Multiple inheritance leads to confusion for the programmer. For example, class A has got a member x and class B has also got a member x. When another class C extends both the classes, then there is a confusion regarding which copy of x is available in C. See the Figure 15.6.

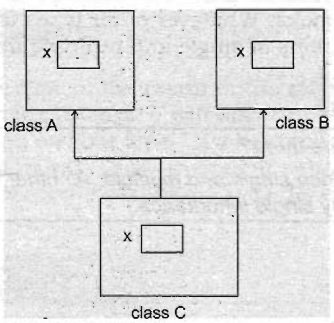


**Figure 15.6** Which copy of x is in class C ?

- Multiple inheritance is available in C++, which is not available in Java. This leads to disappointment in Java programmers. A Java programmer wishes to use multiple inheritance in some cases. Fortunately, JavaSoft people have provided interface concept, expecting the programmers to achieve multiple inheritance by using multiple interfaces. For example, we can write:

```
class Myclass implements interface1, interface2,…
```

In this case, all the members of interface1, interface2,... are accessible in Myclass. This is the way we can achieve multiple inheritance in an indirect way. But this is lacking in real flexibility offered by multiple inheritance. In multiple inheritance, we can use multiple classes where complete methods are available, which can be called and directly used in the sub classes. But in case of interfaces, none of the methods will have method body. So, the programmers should provide body for all the methods of the interfaces in the implementation class, i.e., Myclass. Hence, achieving multiple inheritance by using interfaces is not so useful.

- The programmer feels achieving multiple inheritance by using classes would be practically useful to him. This need is also fulfilled by JavaSoft people. The way to achieve multiple inheritance is by repeating the use of single inheritance. For example,

```
class B extends A  //single inheritance
class C extends B  //single inheritance
```

In the first statement, a copy of A is available in B. In the second statement, a copy of A plus B is available to C. This is nothing but writing:

```
class C extends A,B  //invalid
```

We can not write directly the preceding statement in Java. But, we can write the earlier two statements to implement multiple inheritance.

**Important Interview Question**

*Why multiple inheritance is not available in Java?*

*Multiple inheritance is not available in Java for the following reasons:*

1. *It leads to confusion for a Java program.*
2. *The programmer can achieve multiple inheritance by using interfaces.*
3. *The programmer can achieve multiple inheritance by repeatedly using single inheritance.*

There are only two types of inheritance. But some authors have fancied other types of inheritance and included them also in their books. Whatever other types of inheritance the authors are claiming are none other than the combinations of single and multiple inheritance.

**Important Interview Question**

*How many types of inheritance are there?*

*There are two types of inheritance single and multiple. All other types are mere combinations of these two. However, Java supports only single inheritance.*

# Conclusion

Parents producing the children and children acquiring the qualities of parents have inspired the creators of OOPS to include inheritance concept into Object Oriented approach. Inheritance is a very powerful concept, as it makes programming easy by reusing the available classes in creating new classes. Reusability is achieved because of the sub class object containing a copy of super class object within it. The programmer can use 'super' keyword if he wants to access all the members of super class directly in the subclasses. There are two types of inheritance single and multiple. But in Java, only single inheritance is available. This does not make Java handicapped, because there are ways to achieve multiple inheritance by other means.