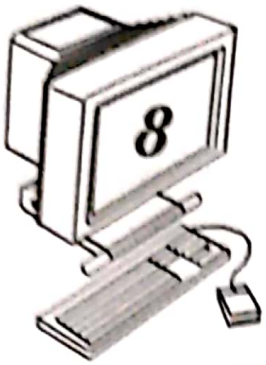


Inheritance and Polymorphism

8.1 Inheritance Basics

A child inherits the property of his parent. He can acquire new properties or modify the inherited one. Similarly, inheritance is a feature of OOP, which allows making use of the existing class without making changes to it. This can be achieved by deriving a new class (Derived class) from the existing class (Base class). A derived class inherits all the properties of the base class. More properties can be added to the derived class, if needed. Therefore, the complexity of the derived class may grow as the level of inheritance grow. There is no limit to the level of inheritance.

Note :

The mechanism of deriving a new class from an old one is called **inheritance**. The old class is called the **base class** or **super class** or **parent class** and the new one is called the **subclass** or **derived class** or **child class**.

For example, scooter is a class in itself. It is also a member of two wheelers class. Two wheelers class in turn is a member of an automotive class as shown in figure 8.1.

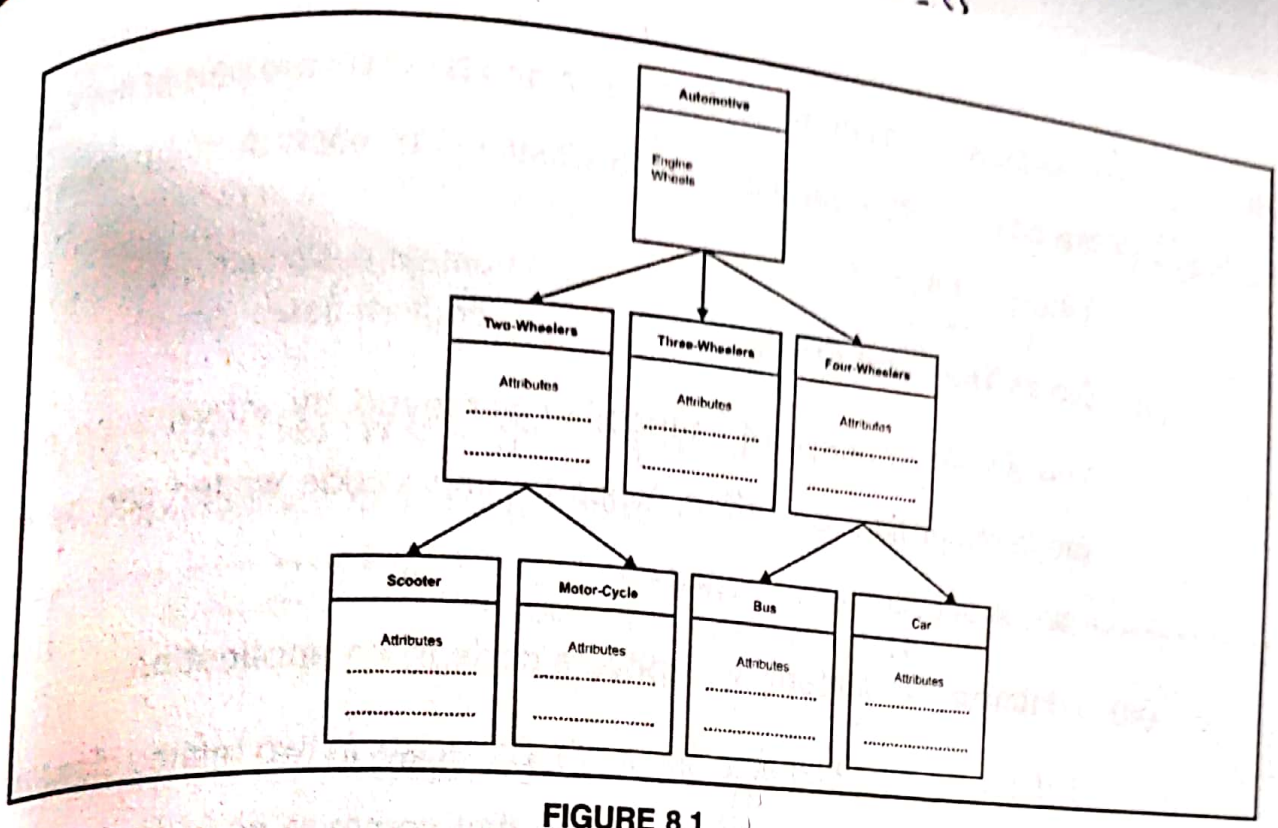


FIGURE 8.1

8.1 Benefits of Inheritance

Main benefits of inheritance are :

(i) **Reusability :** Main advantages of inheritance is **reusability.** Reusability means that we can add additional features of an existing class without modifying it as shown in figure 8.2.

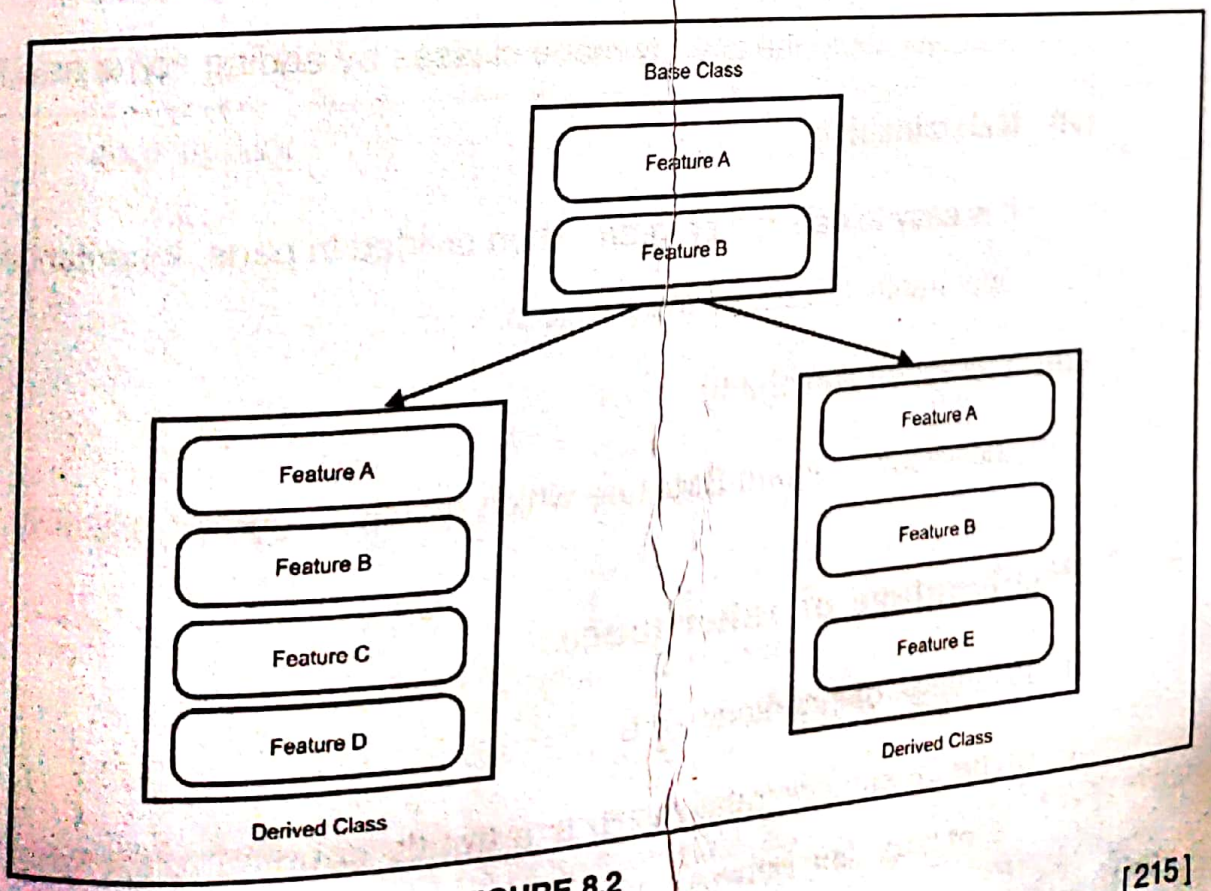


FIGURE 8.2

Note than in the figure 8.2, features A and B, which are part of the base class, are common to both the derived classes, but each derived class also has features of its own.

(ii) Saves Time and Effort

The above concept of reusability achieved by inheritance saves the programmer time and effort. Since the main code written can be reused in various situations as needed.

(iii) Minimize the amount of duplicate code in an application

If duplicate code (variable and methods) exists in two related classes, we can refactored that hierarchy by moving that common code up to the common superclass.

(iv) Better organization of code

Moving of common code to superclass results in better organization of code.

(v) Extendability

We can extend the already made classes by adding some new features.

(vi) Maintainability

It is easy to debug a program when divided in parts. Inheritance provides an opportunity to capture the problem.

(vii) Increased Reliability

Increases Program Structure which results in greater reliability.

8.1.2 Disadvantage of Inheritance

Disadvantage of inheritance are :

(i) The inheritance relationship is a **tightly coupled** relationship, there will be tight bounding between parent and child.

(ii) As classes are tightly coupled, they cannot work independently of each other.

(iii) If we change code of parent class it will get affects to the all the child classes which is inheriting the parent code.

(iv) If super class method is deleted code may not work as subclass may call the super class method.

8.2 Defining Derived Classes (or Subclasses)

The general form of defining a derived class is

```
class derived_class_name extends base_class_name
{
        ...
        ... // members of derived class
        ...
}
```

The keyword **extends** signifies that the properties of the derived class are extended to the base class. Therefore, now the derived class contains all its own members as well as those of the base class.

The definition of derived class is similar to a normal class definition except for the use of extends keyword and base class name.

Example 8.1

```
class student
{
                .....
                .....     // members of base class
                .....
}

class test extends student
{
                .....
                .....     // members of derived class
                .....
}
```

In this example, **test** is the name of derived class and **student** is the name of base class. A derived class gains all the non-private members of its base class.

Example 8.2

Study the following program to understand the concept of inheritances.

```
class base        // Super class
{
        int a, b ;
        void get_ab (int x, int y)
        {
                a = x;
                b = y;
        }
        void add2 ( )
        {
                System.out.println ("a = " + a);
                System.out.println ("b = " + b);
                System.out.println ("Sum of two numbers = " + (a+b));
        }
}       // end of superclass

class derived extends base       // Sub class
{
        int c ;
        void get_c (int z)
        {
                c = z;
        }
        void add3 ( )
        {
                System.out.println ("a = " + a);
                System.out.println ("b = " + b);
                System.out.println ("c = " + c);
                System.out.println ("Sum of three numbers = " + (a+b+c));
        }
}       // end of superclass
```

```
class inheritance
{
        public static void main (String arg [])
        {
                derived d = new derived ( );
                // Accessing base class methods using derived class object
                d.get_ab (10, 20);
                d.add2 ( );

                // Accessing derived class methods
                d.get_c (30);
                d.add3( );
        }
}
```

Output :

```
a = 10
b = 20
Sum of two numbers = 30
a = 10
b = 20
c = 30
Sum of three numbers = 60
```

In derived class, we have not declared variables **a** and **b**, but we have used them in derived class methods. This is possible just because of inheritance. Also it is clear from this example that derived class object can access base class methods as well as derived class methods.

8.3 Access Control

The access control determines how a member can be accessed. The access specifiers provided by java are **public, private, friendly** (or default) and **protected.**

(a) public Access

A variable or method declared as **public** has the widest possible visibility and accessible everywhere. Public member of a class is visible or accessible to its own class, its derived classes and outside the class.

public variable or method can be declared as :

```
public int x ;
public void sum ( ) {.....}
```

(b) friendly Access (default)

If we did not specify the access control modifier, the member has accessibility known as **friendly level of access**. The difference between friendly access and public access is that the public modifier makes the member visible to all classes regardless of their package while friendly access makes members visible only inside the package. (A package is a group of related classes stored separately.)

(c) protected Access

protected modifier makes the members accessible not only to classes and subclasses in the same package but also to subclasses in other packages. Hence we can say the access level lies between public and friendly access.

(d) private

The private members of a class can be accessed only within their own class. They cannot be inherited by subclasses and are therefore inaccessible in subclasses.

(e) private protected Access

In Java, we can use two access control modifiers as

private protected int number ;

The access level provided by such modification is between protected and private access.

Note :

Derived classes do not access to the private member of base classes.

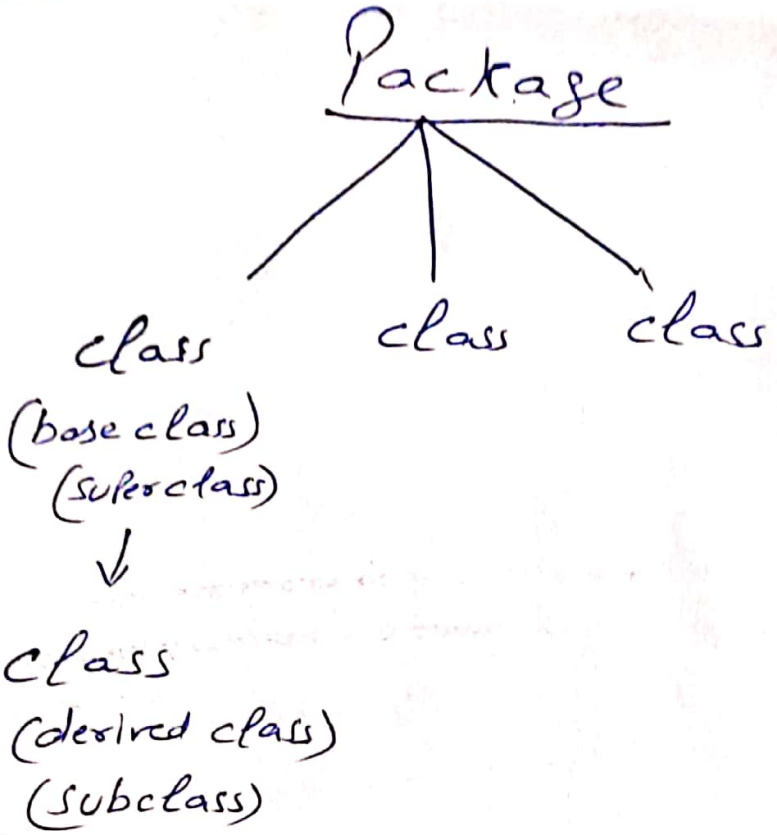
What do we do if the private data needs to be inherited by a derived class? One way to solve this problem is by making the member **public**. Of course, making the member public also makes it available to all other code, which may not be desirable. Fortunately, Java allows you to create a **protected member** in this case.

A protected member is created using the **protected** access modifier. A protected member of the base class becomes a protected member of the derived class and is, therefore, accessible by the derived class. The Table 8.1 shows the visibility provided by various access modifiers.

TABLE 8.1

Access Location ↓ / Access Modifier →	public	protected	friendly (default)	private protected	private
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same packages	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

of Inheritance



A Package is a group of similar type of classes.

Java environment provides a powerful means of grouping related classes and interfaces together in a single unit called Packages.

.4 Types of Inheritance

Java supports three types of inheritance :

1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance

Java does not support multiple inheritance directly because multiple inheritance supports multiple base classes and in java, the classes cannot be inherited from more than one class. However, you can implement multiple inheritance through **interfaces.**

4.1 Single Inheritance

In single in heritance, there is only one base class and one derived class as shown in the figure 8.3.

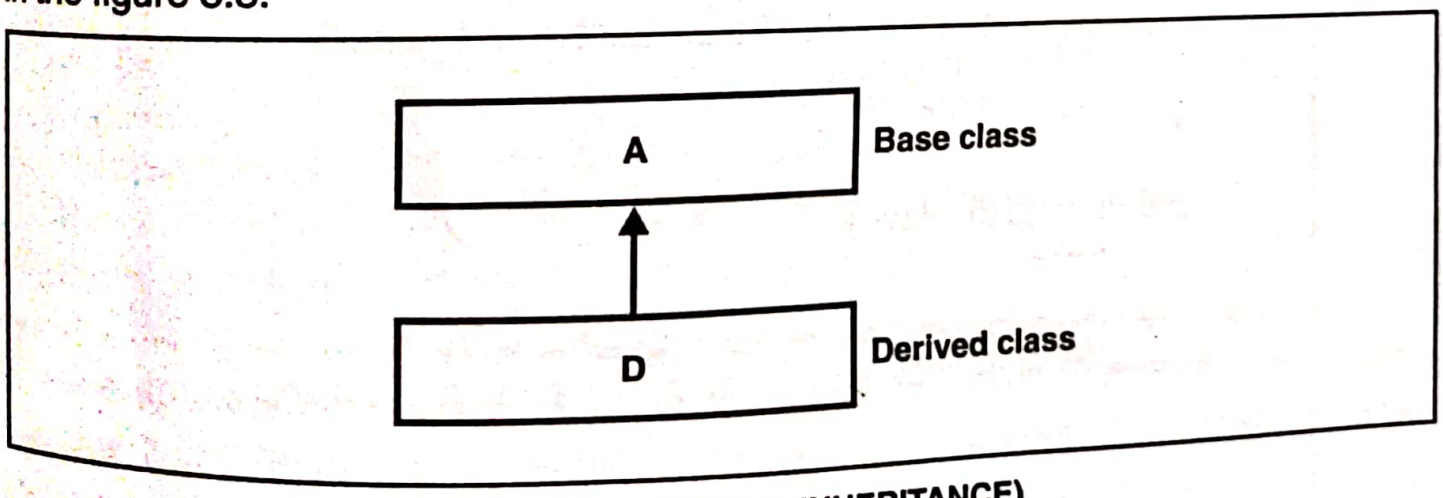


FIGURE 8.3 (SINGLE INHERITANCE)

Example 8.3

Figure 8.4 and following program illustrate single inheritance.

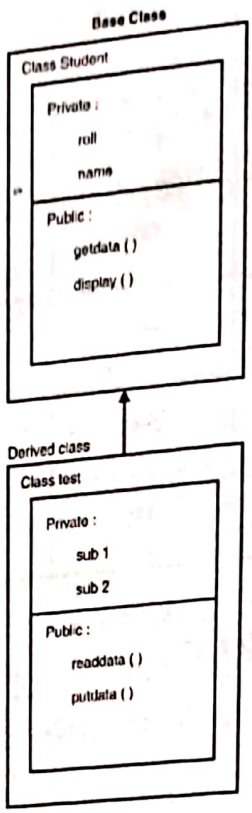


FIGURE 8.4

```
import java.util.Scanner;

//Base class
class student
{
    int roll;
    String name;

    //Create Scanner Object
    Scanner s=new Scanner(System.in);

    public void getdata( )

    {
        System.out.println("Enter Roll No and Name");
        roll = s.nextInt();
        name = s.next();
    }

    public  void display( )

    {
        System.out.println("Roll No = " + roll);
        System.out.println("Name = " +name);
    }
}
```

```
// Derived class
class test extends student
{
    int sub1, sub2;
    public void readdata( )
    {
        getdata( );          //base class method
        System.out.println("Enter the Marks of two subjects");
        sub1 = s.nextInt();
        sub2 = s.nextInt();
    }
    public void putdata( )
    {
        display( );          //base class method
        System.out.println("Marks of subject1 " + sub1);
        System.out.println("Marks of subject2 " + sub2);
    }
}
class singleinheritance
{
    public static void main(String args[])
    {
        test obj = new test( );
        obj.readdata( );
        obj.putdata( );
    }
}
```

Output :

```
Enter Roll No and Name
21
Shivank
Enter the Marks of two subjects
80
85
Roll No = 21
Name = Shivank
Marks of subject1 80
Marks of subject2 85
```

In this example, name of base class is **student** and name of derived class is **test.** It is clear from the program that all the non-private members of base class are the members of derived class.

8.4.2 Multilevel Inheritance

In multilevel inheritance, a class is derived from an already derived class as shown in figure 8.5.

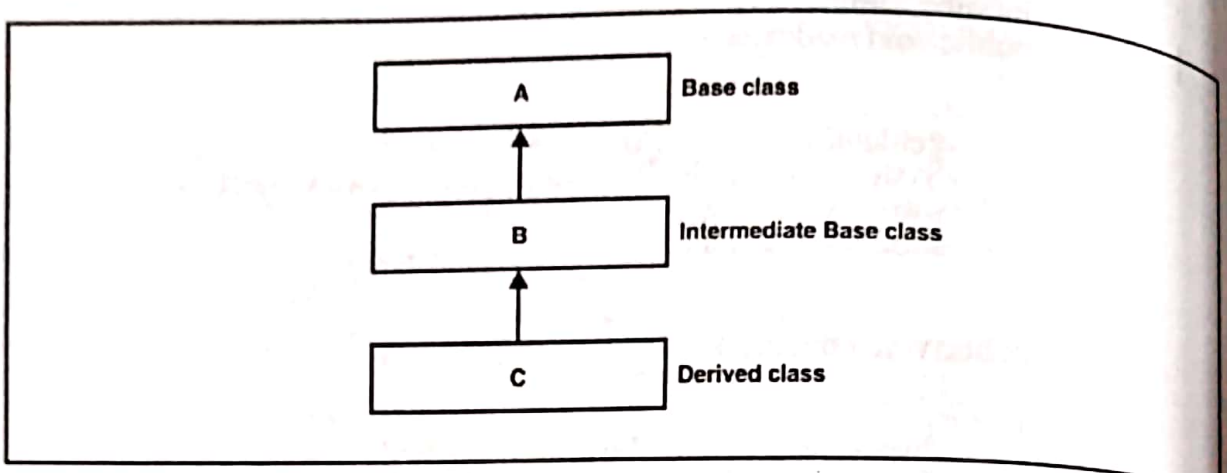


FIGURE 8.5

The class A is a base class for the derived class B. Class B is a base class for the derived class C. Therefore, the class B is known as **intermediate base class.**

Example 8.4

Figure 8.6 and following program illustrate multilevel inheritance.

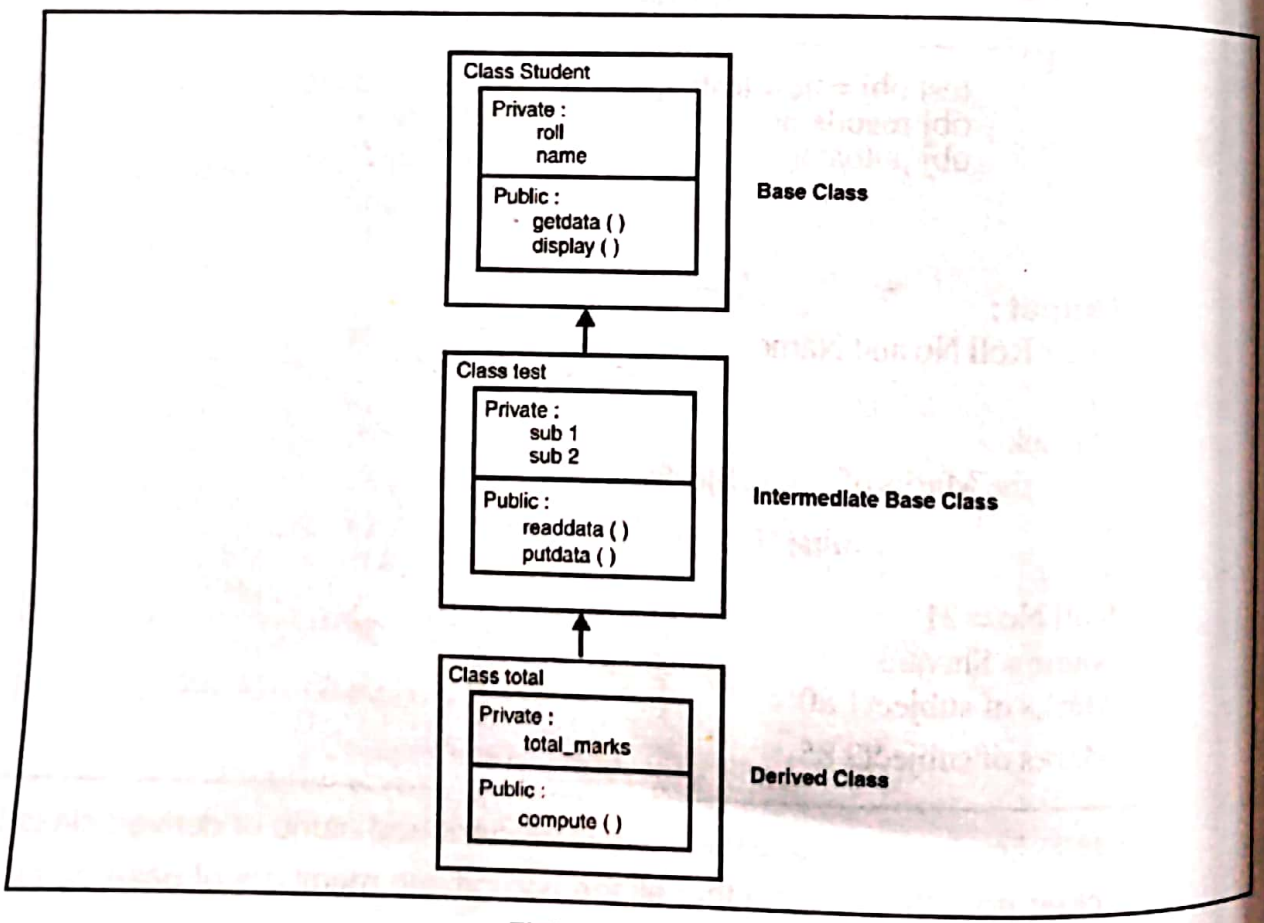


FIGURE 8.6

```
import java.util.Scanner;

//Base class
class student
{
    int roll;
    String name;

    //Create Scanner Object
    Scanner s=new Scanner(System.in);

    public void getdata( )

    {
        System.out.println("Enter Roll No and Name");
        roll = s.nextInt();
        name = s.next();
    }

    public void display( )

    {
        System.out.println("Roll No = " + roll);
        System.out.println("Name = " + name);
    }
}

// Intermediate base class
class test extends student

{
    protected int sub1, sub2; //private will not work
    public void readdata( )

    {
        getdata( );          //base class method
        System.out.println("Enter the Marks of two subjects");
        sub1 = s.nextInt();
        sub2 = s.nextInt();
    }

    public void putdata( )

    {
        display( );          //base class method
        System.out.println("Marks of subject1 " + sub1);
        System.out.println("Marks of subject2 " + sub2);
    }
}
```

```
//Derived class
class total extends test

{

    int total_marks;
    public void compute()

    {
        total_marks = sub1 + sub2;
        System.out.println("Sum of Marks = " + total_marks);
    }
}


class multilevelinheritance

{
    public static void main(String args[])

    {
        total obj = new total( );
        obj.readdata( );
        obj.putdata( );
        obj.compute( );
    }
}
```

Output :

```
Enter Roll No and Name
21
Shivank
Enter the Marks of two subjects
80
85
Roll No =21
Name = Shivank
Marks of Subject1 80
Marks of Subject2 85
Sum of Marks = 165
```

8.3 Hierarchical Inheritance

In hierarchical inheritance, two or more classes are derived from one base class as shown in figure 8.7.

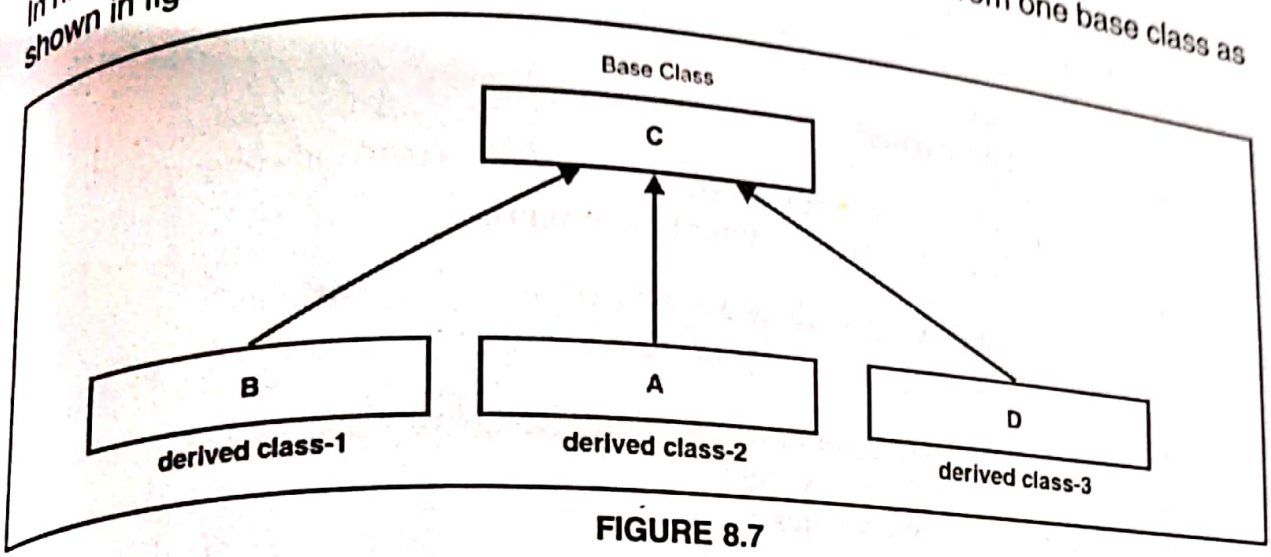
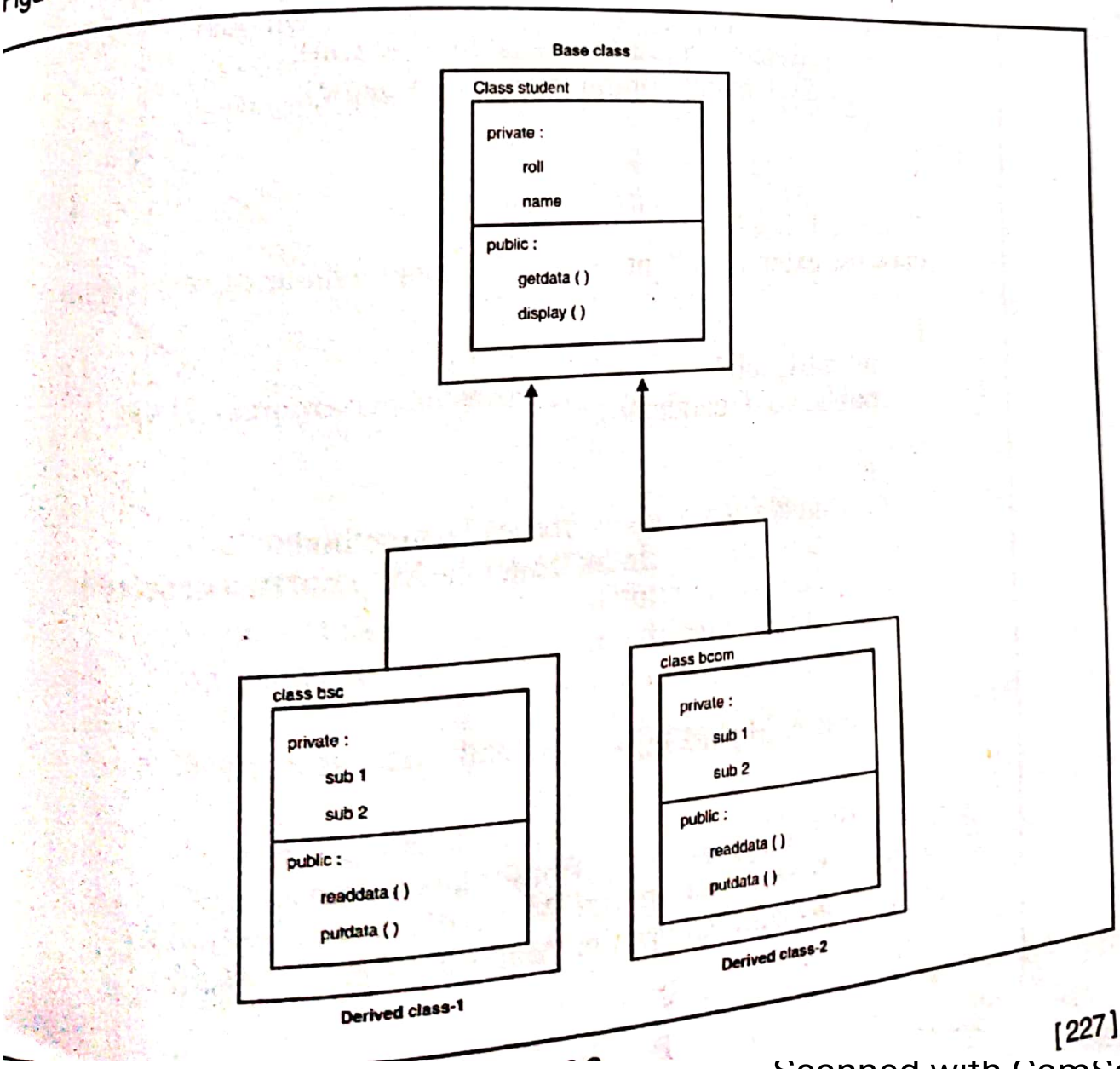


FIGURE 8.7

Example 8.5

Figure 8.8 and following program illustrate the hierarchical inheritance.



```
import java.util.Scanner;

//Base class
class student
{
    int roll;
    String name;

    //Create Scanner Object
    Scanner s=new Scanner(System.in);

    public void getdata( )

    {
        System.out.println("Enter Roll No and Name");
        roll = s.nextInt();
        name = s.next();

    }

    public void display( )

    {
        System.out.println("Roll No = " + roll);
        System.out.println("Name = " + name);

    }
}

// Derived class1
class bsc extends student

{
    int sub1, sub2;
    public void readdata( )

    {
        getdata( );          //base class method
        System.out.println("Enter the Marks of two subjects");
        sub1 = s.nextInt();
        sub2 = s.nextInt();
    }

    public void putdata( )

    {
        display( );          //base class method
        System.out.println("Marks of subject1 " + sub1);
        System.out.println("Marks of subject2 " + sub2);
    }
}
```

```
// Derived class2
class bcom extends student

{
    int sub1, sub2;
    public void readdata( )

    {
        getdata( );          //base class method
        System.out.println("Enter the Marks of two subjects");
        sub1 = s.nextInt();
        sub2 = s.nextInt();
    }

    public void putdata( )

    {
        display( );          //base class method
        System.out.println("Marks of subject1 " + sub1);
        System.out.println("Marks of subject2 " + sub2);
    }
}

class hierarchicalinheritance

{
    public static void main(String args[])

    {
        bsc obj = new bsc( );        //object of derived class1 created
        System.out.println("Enter the data of a B.Sc. Student ");
        obj.readdata( );
        obj.putdata( );

        bcom obj1 = new bcom( ); //object of derived class2 created

        System.out.println( );
        System.out.println("Enter the data of a B.Com. Student ");
        obj1.readdata( );
        obj1.putdata( );
    }
}
```

Output :

```
Enter.the data of a B.Sc. Student
Enter Roll No and Name
21
Shivank
Enter the Marks of two subjects
80
85
Roll No = 21
Name = Shivank
Marks of Subject1 80
Marks of Subject2 85

Enter the data of a B.Com. Student
Enter Roll No and Name
25
Smriti
Enter the Marks of two subjects
84
76
Roll No = 25
Name = Smriti
Marks of Subject1 84
Marks of Subject2 76
```