UNIT 6: INHERITANCE

UNIT STRCUTURE

6.1 Learning Objectives
6.2 Introduction
6.3 Inheritance
6.3.1 Single Inheritance
6.3.2 Multilevel Inheritance
6.4 Final Keyword
6.5 Abstract Class and Method
6.6 Static Members
6.7 Let Us Sum Up
6.8 Further Readings
6.9 Answers to Check Your Progress
6.10 Model Questions

6.1 LEARNING OBJECTIVES

After going though this unit, you will be able to:

- define inheritance in Java
- describe different types of inheritances in java
- describe abstract method and abstract class
- describe static members

6.2 INTRODUCTION

In the previous unit we have learned the concept of arrays, strings and vectors. This unit focuses on inheritance and abstract class. Inheritance is a mechanism in which one object acquires all the properties and behaviors of parent object. With the use of inheritance we can create new classes that are built upon existing lasses. When we inherit from an existing class, we can reuse methods and data members of parent class, and new methods and data members can also be added. Abstract class and static members concepts are also covered in detail in this unit. This is the last unit

of this block. So in the next block we will explore concepts related to packages, I/O in Java, Applets and many more.

6.3 INHERITANCE

Reusability is one of the important feature of object-oriented- programming and it can be achieved through ***inheritance***. Java supports the concepts of inheritance. With the use of inheritance the information is made manageable in a hierarchical order. Inheritance can be defined as the process where one object acquires the properties of another. When we want to create a new class and there is already a class that includes some of the code that we want, we can derive the new class from the existing class. In doing this, we can reuse the fields and methods of the existing class without rewriting them again.

A class that is derived from another class is called a ***subclass*** (also a *derived class*, *extended class*, or *child class*). A subclass inherits all the *members* (fields, methods, and nested classes) from its superclass. The class from which the subclass is derived is called a ***superclass*** (also a *base class* or a *parent class*). Constructors cannot be inherited by subclasses, but the constructor of the superclass can be invoked from the subclass. In Java, inheritance is implemented by the process of extension. To define a new class as an extension of an existing class, we simply use an ***extend*** clause in the header mof the new classes definition. The concept of inheritance is used to make the things from general to more specific.

For example, when we hear the word 'vehicle' then we get an image in our mind that it moves from one place to another and that is used for traveling or carrying goods but the word vehicle does not specify whether it is two or three or four wheeler because it is a general word. But the word car makes a more specific image in mind than vehicle, that the car has four wheels. It concludes from the example that car is a specific word and vehicle is the general word. If we think technically about this example then vehicle is the super class (or base class or parent class) and car is the subclass or child class because every car has the features of its parent (in this case vehicle) class. Now, we are going to describe the types of inheritance supported by Java.

6.3.1 Single Inheritance

When a subclass is derived from its parent class then this mechanism is known as single inheritance. In case of single inheritance there is only a sub class and its parent class. It is also called *one level* inheritance. The pictorial representation of single inheritance is shown in figure 6.1.

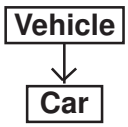


Figure 6.1: Single Inheritance

For example, let us consider a simple example for the demonstration of single inheritance:

//***Program 6.1 :*** Program showing Single Inheritance

```
class A   //  super class A
{
     int x;
     int y;
     int getValue(int p, int q)
     {
          x = p;
          y = q;
          return(0);
     }
     void Show()     {
          System.out.println(x);
     }
}
class B extends A    // subclass B inheriting getValue A
{
     public static void main(String args[ ])    {
          A a = new A();
          a.getValue(5,10);
```

```
            a.Show();
      }
      void display()  {
            System.out.println("I am in B");
      }
}
```

The output will display only 5. The ***getValue()*** and ***show()*** are members of superclass ***A***. With the statement ***a.getValue(5,10);*** the ***getValue( )*** is inherited from class ***A***. As the ***Show()*** method of class ***A*** is displaying only the first parameter so it is displaying only one value in the subclass ***B*** although it is taking two values 5 and 10 when it invoked by the statement ***a.Show()*** in class subclass ***B*** .

6.3.2 Multilevel Inheritance

It is the enhancement of the concept of inheritance. When a subclass is derived from a derived class then this mechanism is known as the *multilevel inheritance*. The derived class is called the subclass or child class for its parent class and this parent class works as the child class for its just above (parent) class. Multilevel inheritance can go up to any number of level. The pictorial representation of multilevel inheritance is as follows:

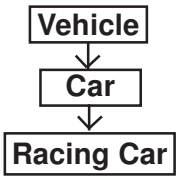


Figure 6.2: Multilevel Inheritance

// ***Program 6.2:*** Program showing Mutilevel Inheritance

```
class A
{
      int x;
      int y;
      int get(int p, int q)
```

```
    {
        x = p;
        y = q;
        return(0);
    }
    void show()
    {
        System.out.println(x);
    }
}
class B extends A //subclass B inheriting from A
{
    void Showb()
    {
        System.out.println("I am in B ");
    }
}
class C extends B //subclass C inheriting from B
{
    void Display()
    {
        System.out.println("I am in C");
    }
    public static void main(String args[])
    {
        A a = new A();
        a.get(5,10);
        a.show();
    }
}
```

The output of the above program will be 5. Here, *a* is an object of superclass *A* and it is inheriting *get( )* and *show( )* methods of *A*. The subclass *B* has one method *Showb()*. The class *C* is the

subclass of *B* and it has one *Display()* method. We can also create objects of class *B* and *C* and use these two method *Showb()* and ***Display()*** for displaying the messages "***I am in B***" and "***I am in C***" respectively.

The mechanism of inheriting the features of more than one base class into a single class is known as *multiple inheritance*. ***Java does not support multiple inheritance.*** But the multiple inheritance can be achieved by using the ***interface***. In Java Multiple Inheritance can be achieved through use of Interfaces by implementing more than one interfaces in a class.

6.4 FINAL KEYWORD

The final keyword is used in several different contexts. Some of them are described below :

final classes : Sometimes we may like to prevent a class from being further subclasses for security reason. If we write the ***final*** keyword infront of a class name in the class declaration then that class cannot be inherited i.e., we cannot create sub classes from that class. This is written as follows:

```
final class A     // A cannot be extended further
      {
                  //statements
      }
```

final methods : We can also declare a method with the final keyword. A method that is declared final cannot be overridden in a subclass. The main reason for declaring a method to be final is to guarabtee that it cannot be changed. For this, we have to just put keyword *final* after the access specifier and before the return type like this:

```
public final String convertText()
        {   //statements
        }
```

final fields : We may also declare fields to be *final*. This is not the same thing as declaring a method or class to be final. When a field is declared

final, it is a constant which will not and cannot change. It can be set once (for instance when the object is constructed, but it cannot be changed after that). Attempts to change it will generate a compile-time error.

final arguments : We can declare a method arguments as *final*. This means that the method will not directly change them. Since all arguments are passed by value, this is not absolutely required, but it is occasionally helpful.

6.5 ABSTRACT CLASS AND METHOD

An ***abstract class*** is a class that has atleast one *abstract method*. An abstract class cannot be instantiated. An ***abstract method*** is not actually implemented in the class. It is merely declared there. The body of the method is then implemented in subclasses of that class. Java allows classes and methods declared to be abstract by means of *abstract modifier*.

Let us consider the following example for the demonstration of abstract class and method. Program 6.3 defines one abstract class ***Shape*** and two general classes ***Circle*** and ***Rectangle***.

```
//Program 6.3 : Program showing use of abstract class
abstract class Shape          //abstract class
{
   abstract double Area();    //abstract method
   abstract double Circumference(); //abstract method
}
//The abstract class Shape has two abstract methods: Area() and
//Circumference(). As abstract methods, they are declared with
//only their prototypes.
class Circle extends Shape    //derive class Circle
{
   double radius;       //field
   Circle(double radius) //constructor
   {
```

```
            this.radius=radius;
        }
        double Circumference()  //method
        {
            return 2*3.14*radius;
        }
        double Area()   //method
        {
            return 3.14*radius*radius;
        }
    }
    //The Circle class has one fields, one constructor and two
methods.
    //The methods Circumference() and Area() implement the corresponding
    // abstract methods declared in the superclass Shape.
    class Rectangle extends Shape//Derive class Rectangle
    {
        double length;      //field for specifying length
        double breadth;     //field for specifying breadth
        Rectangle(double length, double breadth)    //constructor
        {
            this.length=length;
            this.breadth=breadth;
        }
        double Area()       //method
        {
            return length*breadth;
        }
        double Circumference()          //method
        {
            return 2*length*breadth;
        }
    }
    //the Rectangle class has two fields specifying length and
```

```
breadth,
        //one constructor and two methods.The two methods Circumference()
        // and Area() implement the corresponding abstract methods
        // declared in the superclass Shape.
        class DemoAbstract
        {
            public static void main(String[] args)
            {
                 Rectangle r = new Rectangle(10.0,6.0);
                 Circle c = new Circle(4.0);
            System.out.println("The area of the rectangle is = "+ r.Area());
            System.out.println("The circumference of the rectangle is = "+
                           r.Circumference());
            System.out.println("The area of the circle is = "+ c.Area());
            System.out.println("The circumference of the cicle is = "+
                           c.Circumference());
            }
        }
```

The ouput will be like this :

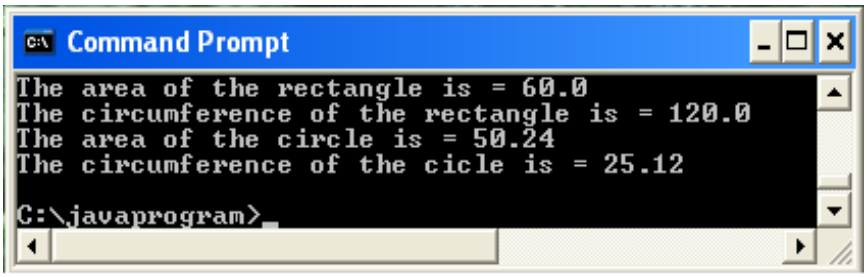


Figure 6.3 : Output of program 6.3

In the program, the abstract ***Area()*** method is declared in the ***Shape*** class above, because we want every subclass to have complete method that returns the areas of its instances and we want all those methods to have the same signature ***double Area()***. Similary for the abstract ***Circumference()*** method.

An abstract method is one that is intended to be *overridden* in each subclass. The abstract method specifies what its subclasses have to implement, but leaves the actual implementation up to them. Thus an abstract method can be regarded as an outline or a specification contact.

6.6 STATIC MEMBERS

There may be some situation where we may want to define a class member that will be used independently without any object of that class. Generally, a class member must be accessed only in conjunction with an object of its class. However, it is possible to create a member that can be used by itself, without reference to a specific instance. To create such a member, we have to precede its declaration with the keyword ***static***. When a member is declared as *static*, it can be accessed before creating any objects of its class are created, and without reference to any object. We can declare both methods and variables to be *static*.

Static methods (***class methods***), like static variables, belong to the class and not to an individual instance of the class. It can be invoked by the method name and through the class name without any object. Because it is not bound to a particular object instance, a static method can directly access only other static members of the class. It cannot directly see any instance variables or call any instance methods.

The most common example of a *static* member is ***main( )***. Method **main( )** is declared as *static* because it must be called before any objects exist. Instance variables declared as *static* are, essentially, global variables. When objects of its class are declared, no copy of a *static* variable is made. Instead, all instances of the class share the same *static* variable. Methods declared as *static* have several restrictions:

- They can only call other *static* methods.
- They must only access *static* data.



CHECK YOUR PROGRESS

Q1. What is the difference between public member and private member of a class?

Q2. What is the difference between class method and an instance method.

Q3. State *True* or *False :*

i) A static (class) method is a method that is invoked without being bound to any specific object of the class.

ii) Protected members are accessible only from within its own class and its extensions.

iii) Private members are accessible only from within its own class.

iv) It is possible to instantiate an abstract class.

v) An abstract method may must be part of an abstract class.

vi) Final classes cannot be inherited.

vii) An abstract class is a class that has atleast one abstract method.

viii) Public members are not accesiible by all classes.



6.7 LET US SUM UP

- We can define one class based on another. This is called *inheritance*. The base class is called a *superclass* and the derived class is called a *subclass*. A superclass can also be a subclass of another superclass.
- The first statement in the body of a constructor for a subclass should call a constructor for the superclass. If it does not, the compiler will insert a call for the default constructor for the superclass.

- Modifiers are some keywords which are used in the declaration of classes, fields, methods, constructors local variables etc. The use of different types of modifiers like *public, private, protected, final, abstract, static* etc. are described in this unit.
- Methods that are specified as static can be called even if no class objects exist, but a static method cannot refer to instance variables.
- An *abstract method* is a method that has no body defined for it, and is declared using the keyword abstract.
- An *abstract class* is a class that contains one or more abstract methods. It must be defined with the attribute abstract.



6.8 FURTHER READINGS

1) Balagurusamy, E. (2007). Programming With Java: A Primer 3E.

2) Schildt, H. (2000). C/C++ Programmer's Reference. McGraw-Hill, Inc..

3) Deitel, P., & Deitel, H. (2011). Java How to program. Prentice Hall Press.



6.9 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1 : A public class member can be accessed from methods of other classes. A private class member can be accessed only from methods of the class.

Ans. to Q. No. 2 : A ***class method*** is declared *static* and is invoked using the class name. For example,

double p = Math.abs(q);

invokes the class ***abs()*** that is defined in the ***Math*** class. An ***instance method*** is declared without the static modifier and is invoked using the name of the object to which it is bound. For example,

double p = **obj**. nextDouble();

invokes the class method *nextDouble()* that is defined in the *Random* class and is bound to the object *obj* which is an instance of that class.

Ans. to Q. No. 3 : i) True; ii) True; iii) True; iv) False; v) True; vi) True; vii) True; viii) False



6.10 MODEL QUESTIONS

Q1. Define is inheritance. How does it help us creating new classes quickly?

Q2. Discuss different levels of access protection available in Java.

Q3. Define inheritance? Explain single inheritance with an appropriate program.

Q4. What do you mean by abstract class? Write an example showing abstract class and abstract methods.

Q5. Design a class to represent a bank account. Include the following data members: *Annount number, Depositor name, Account type, Balance amount in the account* .

Methods : To assign initial value
To deposit an amount
To withdraw an amount after checking balance
To display depositor name and balance.

****