CHAPTER

18

ABSTRACT CLASSES

We know that a class is a model for creating the objects. A class contains description of properties or variables and actions or methods of its objects. If an object has the properties and actions as mentioned in the class, then that object belongs to the class. The rule is that any thing is written in the class is applicable to all of its objects. If a method is written in the class, it is available as it is to all of the class objects. For example, take a class Myclass that contains a method `calculate()` that calculates square value of a given number. If we create three objects to this class, all the three objects get the copy of this method and hence, from any object, we can call and use this method. See Program 1.

Program 1: Write a program where Myclass's `calculate()` method is available to all the objects and hence every object can calculate the square value.

```
//All the objects sharing the same method
class Myclass
{
      //method to calculate square value
      void calculate(double x)
      {
            System.out.println("Square= "+ (x*x));
      }
}
class Common
{
      public static void main(String args[ ])
      {
            //create 3 objects
            Myclass obj1 = new Myclass();
            Myclass obj2 = new Myclass();
            Myclass obj3 = new Myclass();

            //call calculate() method from the objects
            obj1.calculate(3);
            obj2.calculate(4);
            obj3.calculate(5);
      }
}
```

Output:

```
C:\> javac Common.java
C:\> java Common
Square= 9.0
Square= 16.0
```

```
Square= 25.0
```

Of course, in the preceding program, the requirement of all the objects is same, i.e., to calcul square value. But, some times the requirement of the objects will be different and entir dependent on the specific object only. For example, in the preceding program, if the first obj wants to calculate square value, the second object wants the square root value and the third obj wants cube value. In such a case, how to write the `calculate()` method in Myclass ?

Since, calculate() method has to perform three different tasks depending on the object, we can write the code to calculate square value in the body of calculate() method. On the other hand, if write three different methods like `calculate_Square()`, `calculate_Sqrt()`, `calculate_Cube()` in `Myclass`, then all the three methods are available to all the three obje which is not advisable. When each object wants one method, providing all the three does not l reasonable. To serve each object with the one and only required method, we can follow the steps:

1. First, let us write a `calculate()` method in Myclass. This means every object wants calculate something.
2. If we write body for `calculate()` method, it is commonly available to all the objects. So le not write body for `calculate()` method. Such a method is called abstract method. Since, write abstract method in Myclass, it is called abstract class.
3. Now derive a sub class `Sub1` from `Myclass`, so that the `calculate()` method is available to sub class. Provide body for `calculate()` method in Sub1 such that it calculates square va Similarly, we create another sub class `Sub2` where we write the `calculate()` method with t to calculate square root value. We create the third sub class `Sub3` where we write `calculate()` method to calculate cube value. This hierarchy is shown in Figure 18.1.
4. It is possible to create objects for the sub classes. Using these objects, the respective met can be called and used. Thus, every object will have its requirement fulfilled.

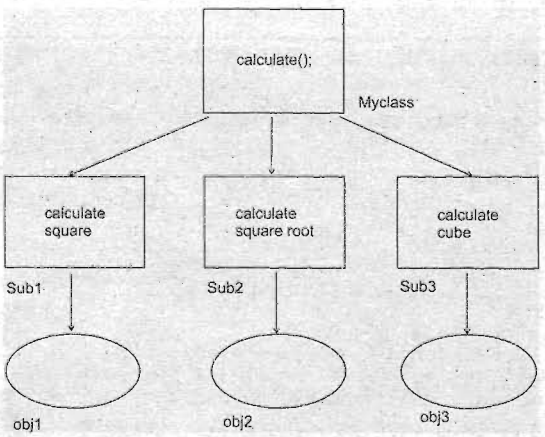

Figure 18.1 Defining a method in sub classes to suit the objects

Abstract Method and Abstract Class

An abstract method does not contain any body. It contains only the method header. So we ca it is an incomplete method. An abstract class is a class that generally contains some ab methods. Both the abstract class and the abstract methods should be declared by using th word 'abstract'.

Since, abstract class contains incomplete methods, it is not possible to estimate the total memory required to create the object. So, JVM can not create objects to an abstract class. We should create sub classes and all the abstract methods should be implemented (body should be written) in the sub classes. Then, it is possible to create objects to the sub classes since they are complete classes.

Important Interview Question

What is abstract method?

An abstract method is a method without method body. An abstract method is written when the same method has to perform different tasks depending on the object calling it.

What is abstract class?

An abstract class is a class that contains 0 or more abstract methods.

In Program 2, we create Myclass as the abstract super class with an abstract method `calculate()`. This method does not have any body within it. Sub1, Sub2 and Sub3 are the three sub classes where the abstract method is implemented as per the requirement of the objects. Since, the same abstract method is implemented differently for different objects, they can perform different tasks.

Program 2: Let us make a program where the abstract class Myclass has one abstract method which has got various implementations in sub classes.

```
//All the objects need different implementations of the same method
abstract class Myclass
{
      //this is abstract method
      abstract void calculate(double x);

}
class Sub1 extends Myclass
{
      //calculate square value
      void calculate(double x)
      {
            System.out.println("Square= "+ (x*x));
      }
}
class Sub2 extends Myclass
{
      //calculate square root value
      void calculate(double x)
      {
            System.out.println("Square root= "+ Math.sqrt(x));
      }
}
class Sub3 extends Myclass
{
      //calculate cube value
      void calculate(double x)
      {
            System.out.println("Cube= "+ (x*x*x));
      }
}

class Different
{
      public static void main(String args[ ])
      {
            //create sub class objects
            Sub1 obj1 = new Sub1();
            Sub2 obj2 = new Sub2();
            Sub3 obj3 = new Sub3();
```

```
                //let the objects call and use calculate() method
                obj1.calculate(3);  //calculate square
                obj2.calculate(4);  //calculate square root
                obj3.calculate(5);  //calculate cube value

        }
}
```

Output:

```
C:\> javac Different.java
C:\> java Different
Square= 9.0
Square root= 2.0
Cube= 125.0
```

We cannot create an object to abstract class, but we can create a reference variable to it, as variable will not take any memory. Once the reference is created, it can be used to refer to objects of sub classes, as shown here:

```
Myclass ref;  //ref is the reference of Myclass
ref = obj1;   //ref is referring to obj1
ref.calculate(3);  //call obj1' s calculate() method
ref = obj2;   //now ref is referring to obj2
ref.calculate(4);  //call obj2's calculate() method
ref = obj3;   //now ref is referring to obj3
ref.calculate(5);  //call obj3's calculate() method
```

Let us take another example to understand the abstract class concept in a better way. We see m cars on the road. These cars are all objects of car class. For example, Maruti, Santro, Benz ar objects of car class. Suppose, we plan to write Car class, it contains all the properties (varia and methods (actions) of any car object in the world. For example, we can write the follo members in Car class:

- **Registration number:** Every car will have a registration number and hence we write this a instance variable in Car class. All cars whether it is Maruti or Santro should have a registr number. It means registration number is a common feature to all the objects. So, it ca written in the Car class.
- **Fuel tank:** Every car will have a fuel tank, opening and filling the tank is an actio represent this action, we can write a method like:

```
void openTank()
```

How do we open and fill the tank? Take the key, open the tank and fill fuel. Let us assume th cars have same mechanism of opening the tank. So, the code representing the opening mecha can be written in `openTank()` method's body. So it becomes a concrete method. A concrete m is a method with body.

- **Steering:** Every car will have a steering and steering the car is an action. For this, we w method as:

```
void steering(int direction, int angle)
```

How do we steer the car? All the cars do not have same mechanism for steering. Maruti cars manual steering. Santro cars have power steering. So, it is not possible to write a part mechanism in `steering()` method. So, this method should be written without body in Car cl becomes an abstract method.

- **Brakes:** Every car will have brakes. Applying brakes is an action and hence it can be represented as a method, as:

```
void braking(int force)
```

How do we apply brakes? All cars do not have same mechanism for brakes. Maruti cars have hydraulic brakes. Santro cars have gas brakes. So, we cannot write a particular braking mechanism in this method. This method, hence, will not have a body in Car class, and hence becomes abstract. See Figure 18.2.

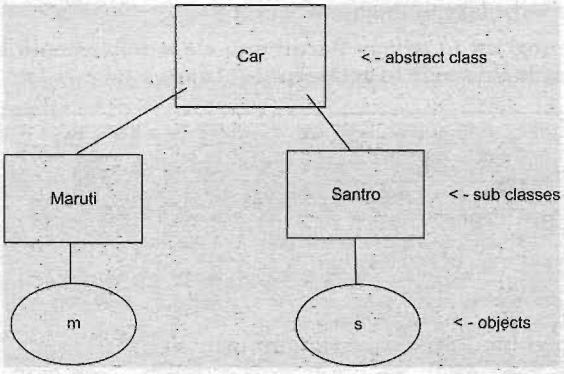

Figure 18.2 Abstract class and its sub classes

So, the Car class has an instance variable, one concrete method and two abstract methods. Hence, Car class will become abstract class. See this class in Program 3 given here.

Program 3: Write a program in which abstract class `Car` contains an instance variable, one concrete method and two abstract methods. Compile this code to get `Car.class`. We can not run it because it does not have a `main()` method.

```
//This is an abstract class
abstract class Car
{
      //every car will have a registration number
      int regno;

      //initialize the value of regno
      Car(int r)
      {
            regno=r;
      }

      //all cars will have a fuel tank and same mechanism to open the tank
      void openTank()
      {
            System.out.println("Fill the tank");
      }

      //all cars will have steering but different cars will have different
      //steering mechanisms.
      abstract void steering(int direction,int angle);

      //all cars will have brakes but different cars will have different
      //braking mechanisms.
      abstract void braking(int force);
}
```

Output:

```
C:\> javac Car.java
C:\>
```

Now, we have written the abstract class, the next step is to derive sub classes from the Car class. I the sub classes, we should take the abstract methods of the Car class and implement (writing bod in) them. The reason why we implement the abstract methods in the sub classes is that th implementation of these methods is dependent on the sub classes. In our program, let us wri Maruti and Santro as the two sub classes where the two abstract methods will be implemente accordingly. See these sub class in Programs 4 and 5.

Program 4: Write a program in which Maruti sub class implements the abstract methods of t super class, `Car`. Compile this code to get `Maruti.class`.

```
//This is a concrete sub class derived from Car class
class Maruti extends Car
{
      //store regno in super class var
      Maruti(int regno)
      {
            super(regno);
      }

      //Maruti uses ordinary steering
      void steering(int direction,int angle)
      {
            System.out.println("Take a turn");
            System.out.println("This is ordinary steering");
      }

      //Maruti uses hydraulic brakes
      void braking(int force)
      {
            System.out.println("Brakes applied");
            System.out.println("These are hydraulic brakes");
      }
}
```

Output:

```
C:\> javac Maruti.java
C:\>
```

Program 5: Write a program in which Santro sub class implements the abstract methods of super class, `Car`. Compile this code to get `Santro.class`.

```
//This is a concrete class derived from class Car
class Santro extends Car
{
      //store regno at super class
      Santro(int regno)
      {
            super(regno);
      }

      //Santro uses power steering
      void steering(int direction,int angle)
      {
            System.out.println("Take a turn");
            System.out.println("This car uses power steering");
      }
```

```
        //Santro uses gas breaks
        void braking(int force)
        {
                System.out.println("Brakes applied");
                System.out.println("This cars uses gas brakes");
        }
}
```

Output:

```
C:\> javac Santro.java
C:\>
```

The next step is to use the sub classes by creating objects. Since, we cannot create an object to `Car` class, we can create a reference to it, as:

```
Car ref;
```

This reference can be used to refer to any of the sub class objects. Through this reference, all features of Maruti and Santro can be used. This can be seen in Program 6.

Program 6: Let us create a program to use all the features of abstract class by creating a reference to it and referring to the sub class objects. Abstract class reference can be used to call the methods of the sub classes.

```
//Using cars
class UseCar
{
        public static void main(String args[ ])
        {
                //create sub class objects
                Maruti m = new Maruti(1001); //1001 is regno.
                Santro s = new Santro(5005); //5005 is regno.

                //create a reference to super class: Car
                Car ref;

                //to use the Maruti car
                ref=m; //to use Santro car: ref=s;

                //use the features of the car
                ref.openTank();
                ref.steering(1,90);
                ref.braking(500);
        }
}
```

Output:

```
C:\> javac UseCar.java
C:\>java UseCar
Fill the tank
Take a turn
This is ordinary steering
Brakes applied
These are hydraulic brakes
```

In the preceding program, if reference is used to refer to Santro object, as:

```
ref = s;
```

Then, the following output can be expected:

```
Fill the tank
Take a turn
This car uses power steering
Brakes applied
This cars uses gas brakes
```

In the preceding program, we created a reference of the super class and using this reference, we accessing all the features of the sub classes in `main()` method. Why should we create a referenc the super class? Can't we access the features of the sub classes by individually creating sub c objects?

It is perfectly possible to access all the members of the sub classes by using sub class objects. we prefer to use super class reference to access the sub class features because, the refer variable can access only those features of the sub classes which have been already declare super class. If we write an individual method in the sub class, the super class reference ca access that method. This is to enforce discipline in the programmers not to add any of their features in the sub classes other than whatever is given in super class. For example, a program has added the following method in Maruti class:

```
void wings()
{
    System.out.println("I can fly");
}
```

This method declaration is not found in super class (`Car`), so super class reference cannot ref this method. When wings are added to Maruti car, there is doubt whether it still belongs to class or not. When wings are there, it may belong to Bird class or Aeroplane class, who knows!

Important Interview Question

How can you force your programmers to implement only the features of your class?

By writing an abstract class or an interface.

We take another program where abstract class can be used practically. Suppose, we are writ program to calculate electricity bills. There are two types of electricity connections, commercia domestic. Depending on the type of connection, the rate per unit will vary. So, we take an ab class Plan with instance variables `rate` and to store rate, we write an abstract method `getRa` Then, we create a sub class `CommercialPlan` where rate per unit is taken as Rs. 5.00 per Another sub class `DomesticPlan` will store Rs. 2.60 per unit as rate. Then, we write a se class with the name Calculate to calculate the electricity bill and display the bill amount. I class, we use abstract class reference to refer to `CommercialPlan` class object to calculate t according to commercial plan. Similarly, we use abstract class reference to refer to `Domesti` class object to calculate the bill according to domestic plan.

Program 7: Let us make a program to write abstract class with an instance variable: r concrete method: `getRate()` and an abstract `method:calculateBill()`.

```
//Calculating electricity bill for commercial and domestic plans
abstract class Plan
{
      //take rate as protected to access it in sub classes
      protected double rate;

      //accept rate into rate variable. Since rate will change
      //depending on plan, we declare abstract method
      public abstract void getRate();
```

```
        //calculate the electricity bill by taking units
        public void calculateBill(int units)
        {
                System.out.print("Bill amount for "+ units + " units: ");
                System.out.println(rate*units);
        }
}

class CommercialPlan extends Plan
{
        //store commercial rate as Rs.5.00 per unit
        public void getRate()
        {
                rate = 5.00;
        }

}

class DomesticPlan extends Plan
{
        //store domestic rate as Rs.2.60 per unit
        public void getRate()
        {
                rate = 2.60;
        }
}

class Calculate
{
        public static void main(String args[])
        {
                //create reference p to abstract class
                Plan p;

                //calculate commercial bill for 250 units
                System.out.println("Commercial connection: ");
                p = new CommercialPlan();  //use reference to refer to sub class
                object
                p.getRate();
                p.calculateBill(250);

                //calculate domestic bill for 150 units
                System.out.println("Domestic connection: ");
                p = new DomesticPlan();  //use reference to refer to sub class
                object
                p.getRate();
                p.calculateBill(150);
        }
}
```

Output:

```
C:\> javac Calculate.java
C:\> java Calculate
Commercial connection:
Bill amount for 250 units: 1250.0
Domestic connection:
Bill amount for 150 units: 390.0
```

Summary

Let us summarize the points on abstract class:

- An abstract class is a class that contains 0 or more abstract methods.
- An abstract class can contain instance variables and concrete methods in addition to abstra methods.
- Abstract class and the abstract methods should be declared by using the key word 'abstract'.
- All the abstract methods of the abstract class should be implemented (body) in its sub classes
- If any abstract method is not implemented, then that sub class should be declared as 'abstra In this case, we cannot create an object to the sub class. We should create another sub class this sub class and implement the remaining abstract method there.
- We cannot create an object to abstract class.
- But, we can create a reference of abstract class type.
- The reference of abstract class can be used to refer to objects of its subclasses.
- The reference of abstract class can not refer to individual methods of its subclasses.
- It is possible to derive an abstract class as a sub class from a concrete super class.
- We cannot declare a class as both abstract and final. For example,

```
abstract final class A    //invalid
```

The key word `abstract` represents an incomplete class which depends on the sub classes fo implementation. Creating sub class is compulsory for abstract class. `final` key word prev inheritance. This means, we cannot create sub class to a final class. So, the key words abstr and `final` are contradictory and hence both can not be used simultaneously for a class.

Important Interview Question

Can you declare a class as abstract and final also?

No. `abstract` *class needs sub classes.* `final` *key word represents sub classes which can not b created. So, both are quite contradictory and cannot be used for the same class.*

Conclusion

Any class will have all concrete methods implemented to perform a particular task. So a class perform only that task and hence it is rigid. Abstract class will make programming more flexib giving scope to write abstract methods. It is possible to implement the abstract methods differ in the sub classes of an abstract class. These different implementations will help the programm perform different tasks depending on the need of the sub classes. Moreover, the common mem of the abstract class are also shared by the sub classes. Thus, abstract class is useful to better and flexible programs. Also, because of several levels followed like abstract class, its classes, sub-sub classes, etc., the programmer can manage the code easily.