CHAPTER 12

CLASSES AND OBJECTS

We know a class is a model for creating objects. This means the properties and actions of the objects are written in the class. Properties are represented by variables and actions of the objects are represented by methods. So a class contains variables and methods. The same variables and methods are also available in the objects because they are created from the class. These variables are also called 'instance variables' because they are created inside the object (instance).

If we take Person class, we can write code in the class that specifies the properties and actions performed by any person. For example, a person has properties like name, age, etc. Similarly a person can perform actions like talking, walking, etc. So, the class Person contains these properties and actions, as shown here:

```
class Person
{
      //properties - instance variables
      String name;
      int age;

      //actions - methods
      void talk()
      {
            System.out.println("Hello Iam "+ name);
            System.out.println("My age is "+ age);
      }
}
```

Observe that the key word 'class' is used to declare a class. After this, we should write the class name. In the class, we write instance variables and methods. See the method: `void talk()`. This method does not return any result (void) and it does not accept any data from us. So we did not declare any variables after the method name. Through this method, the person is talking with us; he is introducing himself to us, as shown here:

```
Hello Iam Raju
My age is 22
```

This is what we are displaying in the method. Writing a class like this is not sufficient. It should be used. To use a class, we should create an object to the class. Object creation represents allotting memory necessary to store the actual data of the variables, i.e., Raju and 22. To create an object, the following syntax is used:

```
    Classname  objectname  =  new  Classname();
```

So, to create Raju object to Person class, we can write as:

```
    Person Raju = new Person();
```

Here, 'new' is an operator that creates the object to Person class, hence the right hand side part of the statement is responsible for creating the object. What about the left side statement, which is:

```
    Person Raju;
```

Here, Person is the class name and Raju is the object name. Raju is actually a variable of Person class. This variable stores the reference number of the object returned by JVM, after creating the object. If Raju is a variable, then what is 'Person'? It is, of course the class name, but class is also a data type. So we can say that Raju is a variable of Person class type.

Object Creation

We know that the class code along with method code is stored in 'method area' of the JVM. When an object is created, the memory is allocated on 'heap'. After creation of an object, JVM produces a unique reference number for the object from the memory address of the object. This reference number is also called hash code number.

To know the hashcode number (or reference) of an object, we can use `hashCode()` method of Object class, as shown here:

- ❑ `Employee e1 = new Employee();` //e1 is reference of Employee object.
- ❑ `System.out.println(e1.hashCode());` //displays hash code stored in e1.

Important Interview Question

What is hash code?

Hash code is a unique identification number allotted to the objects by the JVM. This hash code number is also called reference number which is created based on the location of the object in memory, and is unique for all objects, except for String objects.

How can you find the hash code of an object?

The hashCode() method of 'Object' class in java.lang package is useful to find the hash code of an object.

The object reference (hash code) internally represents heap memory where instance variables ar stored. There would be a pointer (memory address) from heap memory to a special structure locate in method area. In method area, a table is available which contains pointers to static variables an methods. This is how the instance variables and methods are organized by JVM internally at ru time. Please see Figure 12.1.

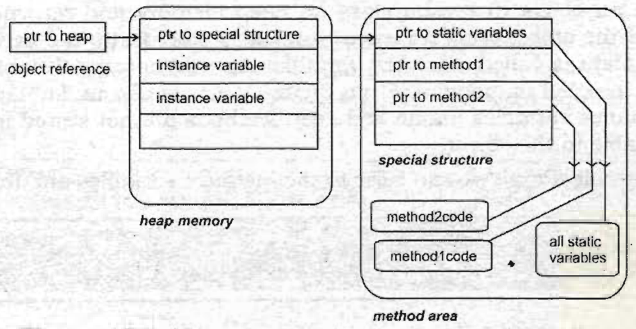

Figure 12.1 Memory organization for members of a class

Program 1: Write a program to create Person class and an object Raju to Person class. Let us display the hash code number of the object, using `hashCode()`.

```
        //Creating a class and object
        class Person
        {
              //properties - variables
              String name;
              int age;

              //actions - methods
              void talk()
              {
                      System.out.println("Hello Iam "+ name);
                      System.out.println("My age is "+ age);
              }
        }
        class Demo
        {
              public static void main(String args[ ])
              {
                      //create Person class object: Raju
                      Person Raju = new Person();

                      //find the hash code of object
                      System.out.println("Hash code= "+Raju.hashCode());
              }
        }
```

Output:

```
        C:\> javac Demo.java
        C:\> java Demo
        Hash code= 1671711
```

In the preceding program, there are two classes. In this case, on which class name we should store the program? The class name which contains `main()` method should be used for this. Since Demo class contains `main()`, JVM starts execution from there and hence the program should be saved as "`Demo.java`".

The hash code displayed by the preceding program may vary from system to system and dependent on the internal memory address by the JVM. Observe the code in Demo class. We created an object to Person class, as:

```
        Person Raju = new Person();
```

Here, JVM will create an object to Person class on heap memory and returns a unique referen number (hash code) of the object. This returned number is stored into the variable Raju, as show in Figure 12.2. Thus, Raju is called reference variable, since by storing the reference number, it referring to the object location in memory. From Figure 12.2, we can understand that the object c heap contains the instance variables (name and age). Methods are not stored in the object on hea But methods are available to the object.

Using the reference variable 'Raju', we can refer to the instance variables and methods, as:

```
Raju.name;
Raju.age;
Raju.talk();
```

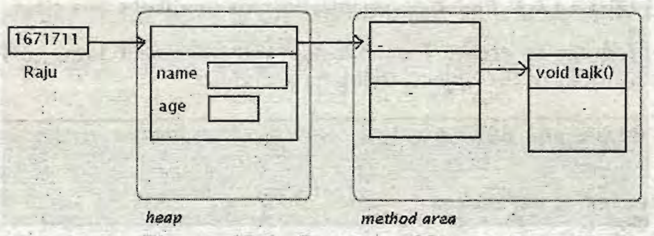

Figure 12.2 Creation of Raju object

Program 2: Let us rewrite the program 1, where we want to call the `talk()` method.

```
//class and object
class Person
{
      //properties - variables
      String name;
      int age;

      //actions - methods
      void talk()
      {
              System.out.println("Hello Iam "+ name);
              System.out.println("My age is "+ age);
      }
}
class Demo
{
      public static void main(String args[ ])
      {
              //create Person class object: Raju
              Person Raju = new Person();

              //call the talk() method
              Raju.talk();
      }
}
```

Output:

```
C:\> javac Demo.java
C:\> java Demo
Hello Iam null
My age is 0
```

Please observe the output of the preceding program. In this program, we created the object 'Raju', but did not initialize the instance variables. Initialization means storing the starting data. Since we did not initialize the instance variables, Java compiler adds some additional code to Person class, as:

```
name = null;
age = 0;
```

When JVM executes the preceding code, it stores null into name and 0 into age. 'null' represents 'nothing'. Table 12.1 summarizes the default values of the instance variables as used by Java compiler.

Table 12.1

Data type	Default value
`byte`	0
`short`	0
`int`	0
`long`	0
`float`	0.0
`double`	0.0
`char`	a space
`String`	null
`any class type`	null
`boolean`	false

Initializing the Instance Variables

It is the duty of the programmer to initialize the instance variables, depending on his requirements. There are various ways to do this. First way is to initialize the instance variables of Person class in the other class, i.e., Demo class. For this purpose, the Demo class can be rewritten like this:

```
class Demo
{
      public static void main(String args[ ])
      {
              //create Person class object: Raju
              Person Raju = new Person();

              //initialize the instance variables using the reference
              Raju. name = "Raju";
              Raju.age = 22;

              //call the talk() method
              Raju.talk();
      }
}
```

```
    private String name = "Venkat";
    private int age = 30;
```

'private' key word does not allow name and age to be accessed by any other class or program from outside. Thus, it protects data. We also use 'public' access specifier for methods as this 'public' key word allows the methods to be accessed by any outside program. Since programmers want to access the methods to perform various tasks, we declare the methods as 'public'. 'private' and 'public' are called 'access specifiers'.

Access Specifiers

An access specifier is a key word that specifies how to access the members of a class or a class itself. We can use access specifiers before a class and its members. There are four access specifiers available in Java:

- **private:** 'private' members of a class are not accessible any where outside the class. They are accessible only within the class by the methods of that class.
- **public:** 'public' members of a class are accessible every where outside the class. So any other program can read them and use them.
- **protected:** 'protected' members of a class are accessible outside the class, but generally, within the same directory.
- **default:** If no access specifier is written by the programmer, then the Java compiler uses a 'default' access specifier. 'default' members are accessible outside the class, but within the same directory.

We generally use private for instance variables, and public for methods. In Java, classes cannot be declared by using 'private'.

Important Interview Question

Can you declare a class as 'private'?

No, if we declare a class as private, then it is not available to Java compiler and hence a compile time error occurs. But, inner classes can be declared as private.

You may feel public, protected and default – all are same. But there is some difference between them. We reserve a complete discussion on access specifiers till the chapter on 'Packages' in Chapter 20.

If the instance variables of Person class are declared as private, then they are not available in Demo class to be initialized. Then how can we initialize them? There is another way. Let us initialize them directly within the Person class at the time of their declaration, as:

```
    private String name = "Raju";
    private int age = 30;
```

But, the problem in this way of initialization is that all the objects are initialized with same data. Please observe Program 4, where we are creating two objects and both are initialized with same data.

Program 4: Write a program to initialize the instance variables directly within the Person class.

```
    //Initializing the instance variables at the time declaration
    class Person
    {
        //instance variables are initialized here
        private String name = "Raju";
        private int age = 30;

        //methods
        void talk()
        {
                System.out.println("Hello Iam "+ name);
                System.out.println("My age is "+ age);
        }
}

class Demo
{
        public static void main(String args[ ])
        {
                //create Person class object: Raju
                Person Raju = new Person();

                //call the talk() method
                Raju.talk();

                //create another Person class object: Sita
                Person Sita = new Person();

                //call the talk() method
                Sita.talk();

        }
}
```

Output:

```
C:\> javac Demo.java
C:\> java Demo
Hello Iam Raju
My age is 30
Hello Iam Raju
My age is 30
```

Please observe that we are creating two objects, 'Raju' and 'Sita'. But both the objects are initialize with same data, Raju and 30. We need Sita object should get Sita's data, not Raju's data.

This way of initialization is suitable for declaring constants. The reason is that constants' value wi not change even though we create several objects. A constant is like a variable, but its value is fixe and cannot be changed. To declare constants, we use the key word 'final', as:

```
final double PI = 3.14159;
```

Since constants are written by using all capitals letters, PI is written in all caps and the final ke word indicates that PI value is fixed. When PI is used any where in the rest of the program, Ja compiler simply substitutes the value 3.14159 in the place of PI.

Constructors

The third possibility of initialization is using constructors. A constructor is similar to a method th is used to initialize the instance variables. The sole purpose of a constructor is to initialize th instance variables. A constructor has the following characteristics:

- The constructor's name and class name should be same. And the constructor's name shoul end with a pair of simple braces.

For example, in Person class, we can write a constructor as:

```
Person()
{
}
```

- A constructor may have or may not have parameters. Parameters are variables to receive data from outside into the constructor. If a constructor does not have any parameters, it is called 'default constructor'. If a constructor has 1 or more parameters, it is called 'parameterized constructor'. For example, we can write a default constructor as:

```
Person()
{
}
```

And a parameterized constructor with two parameters, as:

```
Person(String s, int i)
{
}
```

- A constructor does not return any value, not even 'void'. Recollect, if a method does not return any value, we write 'void' before the method name. That means the method is returning 'void' which means 'nothing'. But in case of a constructor, we should not even write 'void' before the constructor.
- A constructor is automatically called and executed at the time of creating an object. While creating an object, if nothing is passed to the object, the default constructor is called and executed. If some values are passed to the object, then the parameterized constructor is called. For example, if we create the object as:
  - `Person Raju = new Person(); //`here default constructor is called,
  - `Person Raju = new Person("Raju", 22); //`here parameterized constructor will receive "Raju" and 22.
- A constructor is called and executed only once per object. This means when we create an object, the constructor is called. When we create second object, again the constructor is called second time.

Important Interview Question

When is a constructor called, before or after creating the object?

A constructor is called concurrently when the object creation is going on. JVM first allocates memory for the object and then executes the constructor to initialize the instance variables. By the time, object creation is completed, the constructor execution is also completed.

Program 5: Rewrite the previous program by using a default constructor to initialize the instance variables of Person class.

```
//Initializing the instance variables using a default constructor
class Person
{
	//instance variables
	private String name;
	private int age;

	//default constructor
	Person()
	{
```

```
                name = "Raju";
                age = 22;
        }

        //method
        void talk()
        {
                System.out.println("Hello Iam "+ name);
                System.out.println("My age is "+ age);
        }
}

class Demo
{
        public static void main(String args[ ])
        {
                //create Person class object: Raju
                Person Raju = new Person();

                //call the talk() method
                Raju.talk();

                //create another object: Sita
                Person Sita = new Person();

                //call the talk() method
                Sita.talk();

        }
}
```

Output:

```
C:\> javac Demo.java
C:\> java Demo
Hello Iam Raju
My age is 22
Hello Iam Raju
My age is 22
```

From the output, we can understand that the same data "Raju" and 22 are stored in both objects Raju and Sita. Sita object should get Sita's data, not Raju's data. Isn't it? To mitigate problem, let us try parameterized constructor, which accepts data from outside and initializes instance variables with that data. This is what we did in Program 6.

Program 6: Write a program to initialize the instance variables of Person class, using paramete constructor.

```
//Initializing the instance variables using a parameterized constructor
class Person
{
        //instance variables
        private String name;
        private int age;

        //default constructor
        Person()
        {
                name = "Raju";
                age = 22;
        }

        //parameterized constructor
        Person(String s, int i)
        {
                name = s;
                age = i;
        }

        //method
        void talk()
        {
                System.out.println("Hello Iam "+ name);
                System.out.println("My age is "+ age);
        }
}

class Demo
{
        public static void main(String args[ ])
        {
                //create Raju object. Here default constructor is called.
                Person Raju = new Person();

                //call the talk() method
                Raju.talk();

                //create Sita object. Here parameterized constructor is called.
                Person Sita = new Person("Sita", 20);

                //call the talk() method
                Sita.talk();
        }
}
```

Output:

```
C:\> javac Demo.java
C:\> java Demo
Hello Iam Raju
My age is 22
Hello Iam Sita
My age is 20
```

In the preceding program, we achieved our target of initializing the objects with different data. Observe the following code:

```
    //parameterized constructor
    Person(String s, int i)
    {
     name = s;
     age = i;
    }
```

This is a parameterized constructor with two parameters, String s and int i. These parameters are useful to receive data from outside. So while creating the object, we are supposed to pass a string and an integer number, as:

```
Person Sita = new Person("Sita", 20);
```

Now, "Sita" is copied into the first parameter s, and 20 is copied into the second parameter i. From there, these values are copied into the instance variables, name and age in the parameterized constructor. So in Sita, object receives the data, "Sita" and 20.

When we do not pass any values at the time of creating an object, then the default constructor is called. For example, when the object is created as:

```
Person Raju = new Person();
```

Here, the following default constructor is called:

```
    //default constructor
    Person()
    {
          name = "Raju";
          age = 22;
    }
```

And hence, Raju object is initialized with "Raju" and 22.

Important Interview Question

What is the difference between default constructor and parameterized constructor?

Please observe the Table 12.2 given here.

Table 12.2

Default constructor	***Parameterized constructor***
Default constructor is useful to initialize all objects with same data.	*Parameterized constructor is useful to initialize each object with different data.*
Default constructor does not have any parameters.	*Parameterized constructor will have 1 or more parameters.*
When data is not passed at the time of creating an object, default constructor is called.	*When data is passed at the time of creating an object, parameterized constructor is called.*

What is the difference between a constructor and a method?

Please see the Table 12.3 given here.

Table 12.3

Constructors	***Methods***
A constructor is used to initialize the instance variables of a class.	*A method is used for any general purpose processing or calculations.*
A constructor's name and class name should be same.	*A method's name and class name can be same or different.*
A constructor is called at the time of creating the object.	*A method can be called after creating the object.*
A constructor is called only once per object.	*A method can be called several times on the object.*
A constructor is called and executed automatically.	*A method is executed only when we call it.*

Please observe Program 6, where we have written two constructors. Both the constructors have same name, but there is a difference in the parameters. This is called constructor overloading.

Important Interview Question

What is constructor overloading?

Writing two or more constructors with the same name but with difference in the parameters is called constructor overloading. Such constructors are useful to perform different tasks.

When the programmer does not write any constructor in a class, then Java compiler writes a default constructor and initializes the instance variables with the default values which are shown in Table 12.1. This is the reason, we got null and 0 in the output of Program 2.

Let us write another program to understand how to use classes and objects. In this program, we take Person class with name and age. Then we accept the name and age from keyboard with the help of `BufferedReader` in `accept()` method. We check the age in `check()` method. `BufferedReader` belongs to `java.io` package. `readLine()` method of `BufferedReader` class can give `IOException` which we do not want to handle. Since we are calling `readLine()` method in `accept()` method, we should attach '`throws IOException`' after `accept()` method, and since we are calling `accept()` from `main()` method, we should again attach '`throws IOException`' after `main()` method.

Program 7: Write a Java program to understand the use of methods in a class.

```
/* To accept a person's name and age and display if he is
     young, middle aged or old
*/
import java.io.*;
class Person
{
 //instance variables
 private String name;
 private int age;

 //to accept the name and age
 public void accept() throws IOException
 {
      //to accept data from keyboard
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

      //accept name and age
      System.out.print("Enter name: ");
      name = br.readLine();

      System.out.print("Enter age: ");
      age = Integer.parseInt(br.readLine());
 }

 //to check the age and display he is young, middle aged or old
 public void check()
 {
      if(age<=30)
      System.out.println(name+" is young");
      else if(age<=50)
      System.out.println(name+" is middle aged");
      else System.out.println(name+" is old");
 }
}
class Demo
{
      public static void main(String args[ ]) throws IOException
      {

               //create Person class object
               Person p = new Person();

               //accept person data
               p.accept();

               //check the age
               p.check();
      }
}
```

Output:

```
C:\> javac Demo.java
C:\> java Demo
Enter name: Vijay
Enter age: 32
Vijay is middle aged
```

In this program, we can observe that the instance variables of Person class are available within that class to the methods `accept()` and `check()`.

Let us rewrite the preceding program, this time using a parameterized constructor to initialize the instance variables. In this program, we accept the name (s) and age (i) of the person from command line arguments and send them to the constructor at the time of creating the object, as:

```
Person p = new Person(s,i);
```

Here, the parameterized constructor will receive the data, s and i and store them into the instance variables.

Program 8: Write a program to accept a person's name and age through command line arguments and display if he is young, middle aged or old - version 2.

```
//Program 7 rewritten using constructor and command line arguments
import java.io.*;
class Person
{
      //instance variables
      private String name;
      private int age;

      //parameterized constructor
      Person(String s, int i)
      {
              name = s;
              age = i;
      }

      //to check the age and display he is young, middle aged or old
      public void check()
      {
              if(age<=30)
              System.out.println(name+" is young");
              else if(age<=50)
              System.out.println(name+" is middle aged");
              else System.out.println(name+" is old");
      }
}
class Demo
{
      public static void main(String args[ ]) throws IOException
      {
      //create BufferedReader object
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

      //accept person name and age from command line arguments
      //name and age are stored in args[0] and args[1] as strings
      String s = args[0];
      int i = Integer.parseInt(args[1]);

      //create Person class object and pass name and age to the constructor
      Person p = new Person(s,i);

      //check the age
```