CHAPTER

11

INTRODUCTION TO OOPs

The languages like C, Pascal, Fortran etc., are called Procedure oriented programming languages since in these languages, a programmer uses procedures or functions to perform a task. When the programmer wants to write a program, he will first divide the task into separate sub tasks, each of which is expressed as a function. So a C program generally contains several functions which are called and controlled from a main() function. This approach is called *Procedure oriented approach*. Please see Figure 11.1.

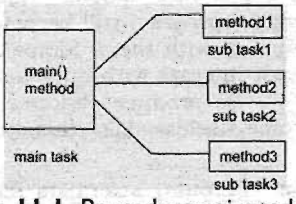

Figure 11.1 Procedure oriented approach

On the other hand, languages like C++ and Java use classes and objects in their programs and are called Object Oriented Programming languages. A class is a module which itself contains data and methods (functions) to achieve the task. The main task is divided into several modules, and these are represented as classes. Each class can perform some tasks for which several methods are written in a class. This approach is called *Object oriented approach*. See Figure 11.2.

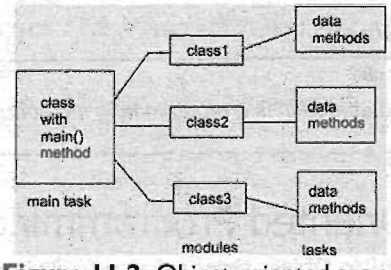

Figure 11.2 Object oriented approach

Programmers have firmly followed Procedure oriented approach for several decades, but as experience and observation teaches new lessons, there is a sudden shift in the software industry for a new approach, called Object oriented approach. First of all, let us discuss the point, why one should go for Object oriented approach when already one has Procedure oriented approach?

Problems in Procedure Oriented Approach

In procedure oriented approach, the programmer concentrates on a specific task, and writes a set of functions to achieve it. When there is another task to be added to the software, he would be writing another set of functions. This means his concentration will be only on achieving the tasks. He perceives the entire system as fragments of several tasks. Whenever he wants to perform a new task, he would be writing a new set of functions. Thus, there will be no reusability of previous functions, in most of the cases. If the programmer can construct the new modules with the help of old modules, i.e., reusing the old modules, programming will become easy. With this view, computer scientists have thought about developing a new approach.

When the software is developed, naturally code size will also be increased. It has been observed in most of the softwares developed following procedure oriented approach, when the code size exceeds 10,000 lines and before reaching 100,000 lines, suddenly at a particular point, the programmers were losing control on the code. This means, the programmers could not understand the exact behavior of the code and could neither debug it, nor extend it. This posed many problems, especially when the software was constructed to handle bigger and complex systems. For example, to create software to send satellites into the sky and control their operations from the ground stations, we may have to write millions of lines of code. In such systems, procedure oriented approach fails and we need another approach.

There is another problem with procedure oriented approach. Programming in this approach is far way from human being's life, and hence considered to be unnatural. Any thing that is not natural leads to certain problems. For example, take the activity, talking. A man talks in his own voice effortlessly. It does not need any special effort on his part. But if the man is asked to imitate the speech of some body else, he feels it is very difficult. Talking in some body else's tone is not a natural activity. Similarly, developing a program will become easy when it is natural. For example, in our life, we have friends. We interact with them. Similarly, a code will have friendly code. Both interact with each other. We will not interact with our enemies. Similarly a code will have enemy code and does not interact with it. We produce children, become aged and die. Similarly code produces new code and may become useless. Like this, programming should stem from human being's life.

Because of the preceding reasons, computer scientists felt the need of a new approach where programming will have several modules. Each module represents a 'class' and the classes can be reusable and hence maintenance of code will become easy. This approach is suitable not only to develop bigger and complex applications but also to manage them easily. Moreover, this approach is built from a single root concept 'object', which represents any thing that physically exists in this world. It means all human beings are objects. All animals are objects. All existing things will become objects. This new approach is called 'Object oriented approach'. Programming in this approach is called 'Object Oriented Programming System (OOPS).

Important Interview Question

What is object oriented approach?

Object oriented programming approach is a programming methodology to design computer programs using classes and objects.

Features of Object Oriented Programming System (OOPS)

There are many features related to Object Oriented approach is discussed in the chapter. Some of the features are:

- Class/Object
- Encapsulation
- Abstraction

- Inheritance
- Polymorphism

Let us move further to have clear understanding of each feature of Object Oriented Approach.

Class/object

Entire OOP methodology has been derived from a single root concept, called 'object'. An object is anything that really exists in the world and can be distinguished from others. This definition specifies that every thing in this world is an object. For example, a table, a ball, a car, a dog, a person, etc., every thing will come under objects. Then what is not an object? If something does not really exist, then it is not an object. For example, our thoughts, imagination, plans, ideas etc., are not objects, because they do not physically exist.

Every object has properties and can perform certain actions. For example, let us take a person whose name is 'Raju'. Raju is an object because he exists physically. He has properties like name, age, sex, etc. These properties can be represented by variables in our programming. For example,

```
String name;
int age;
char sex;
```

Similarly, Raju can perform some actions like talking, walking, eating and sleeping. We may not write code for such actions in programming. But, we can consider calculations and processing of data as actions. These actions are performed by methods (functions). So an object contains variables and methods.

It is possible that some objects may have similar properties and actions. Such objects belong to same category called a 'class'. For example, not only Raju, but also Ravi, Sita, Vijay, etc., persons have same properties and actions. So they are all objects of same class, 'Person'. Now observe that the 'Person' will not exist physically but only Raju, Ravi, Sita etc. exist physically. This means, a class is a group name and does not exist physically, but objects exist physically. See Figure 11.3.

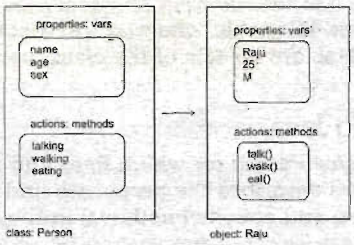

Figure 11.3 Person class and Raju object

To understand a class, take a pen and paper and write down all the properties and actions of any person. That paper contains a model that depicts a person. So it is called a class. We can find a person with the name 'Raju', who got all the properties and actions written on the paper. So 'Raju' becomes an object of the class, Person. This gives a definition for the class. A class is a model or blueprint for creating objects. By following the class, one can create objects. So we can say, whatever is there in the class, will be seen in its objects also.

Important Interview Question

What is the difference between a class and an object?

A class is a model for creating objects and does not exist physically. An object is any thing that exists physically. Both the class and objects contain variables and methods.

We can use a class as a model for creating objects. To write a class, we can write all th characteristics of objects which should follow the class. These characteristics will guide us to creat the objects. A class and its objects are almost the same with the difference that a class does nc exist physically, while an object does. For example, if we say *dog*; it forms a picture into our min with 4 legs, 2 ears, some length and height. This picture in our mind is *class*. If we tally this pictur with the physical things around us, we can find Tom living in our house is satisfying these qualities So Tom, which physically exists, is an object and not a class.

Let us take another example. Flower is a class but if we take Rose, Lily, Jasmine – they are a objects of *flower* class. The class *flower* does not exist physically but its objects, like Rose, Lily Jasmine exist physically.

Let us take another example. We want some shirts stitched by a tailor. First of all, the tailor take the measurements and makes a plan for the shirt, according to the measurements. He may als draw a model shirt in his note book. This plan or model is called a 'class'. Following this model, h stitches the shirts which we can wear. These shirts are called 'objects'. To stitch the shirts, we nee the material 'cloth'. The cloth represents the memory allotted by the JVM for the objects Remember, objects are created on heap memory by JVM at run time. See Figure 11.4. It is als possible to create several objects from the same class. An object cannot exist without a class. But class can exist without any object.

We can think that a class is a model and if it physically forms, then it becomes an object. So a object is called 'instance' (the thing physically happens) of a class.

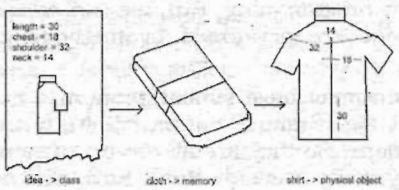

Figure 11.4 Creation of a class and object

Let us take some more examples, like table, chair, cot and sofa are objects of the class, furnitur Similarly, Maruti, Santro and Benz are objects of the class, car. Red, Blue and Green are objects the class, color.

Creating Classes and Objects in Java

Let us create a class with the name Person for which Raju and Sita are objects. A class is created l using the key word, class. A class describes the properties and actions performed by its objects. S we write the properties (variables) and actions (methods) in the class, as:

```
class Person
{
      //properties of a person - variables
      String name;
      int age;

      //actions done by a person - methods
      void talk()
      { }
      void eat()
      { }
}
```

Observe the preceding code. Person class has two variables and two methods. This class code stored in JVM's method area. When we want to use this class, we should create an object to tl class, as:

```
                System.out.println("My age is "+ age);
        }
}
```

In preceding code, the variables name and age are declared as 'private'. There is no way to manipulate them. Only the public method `talk()` can access them. By calling the method `talk()` using Person class object, we can get the person talk with us.

Abstraction

There may be a lot of data, a class contains and the user does not need the entire data. The user requires only some part of the available data. In this case, we can hide the unnecessary data from the user and expose only that data that is of interest to the user. This is called abstraction.

A good example for abstraction is a car. Any car will have some parts like engine, radiator, mechanical and electrical equipment etc. The user of the car (driver) should know how to drive the car and does not require any knowledge of these parts. For example driver is never bothered about how the engine is designed and the internal parts of the engine. This is why, the car manufacturers hide these parts from the driver in a separate panel, generally at the front.

The advantage of abstraction is that every user will get his own view of the data according to his requirements and will not get confused with unnecessary data. A bank clerk should see the customer details like account number, name and balance amount in the account. He should not be entitled to see the sensitive data like the staff salaries, profit or loss of the bank, interest amount paid by the bank and loans amount to be recovered etc,. So such data can be abstracted from the clerk's view. Where as the bank manager is interested to know this data, it will be provided to the manager. Here is an example for abstraction in Java:

```
class Bank
{
        private int accno;
        private String name;
        private float balance;
        private float profit;
        private float loan;

        public void display_to_clerk()
        {
                System.out.println("Accno= "+ accno);
                System.out.println("Name= "+ name);
                System.out.println("Balance= "+ balance);
          }
}
```

In the preceding class, inspite of several data items, the `display_to_clerk()` method is able to access and display only the accno, name and balance values. It can not access profit and loan of the customer. This means the profit and loan data is hidden from the view of the bank clerk.

Inheritance

It creates new classes from existing classes, so that the new classes will acquire all the features o the existing classes is called Inheritance. A good example for Inheritance in nature is parents producing the children and children inheriting the qualities of the parents.

Let us take a class A with some features (members i.e., variables and methods). If we feel another class B wants almost same features, then we can derive or create the class B from A, as:

```
class B extends A
{
}
```

Now, all the features of A are available to B. If an object to B is created, it contains all the members of classes A and also its own members. Thus, the programmer can access and use all the members of both the classes A and B. Thus, class B becomes more useful. This is called inheritance. The original class (A) is called the *super class* and the derived class (B) is called the *sub class*.

There are three advantages of inheritance. First, we can create more useful classes needed by the application (software). Next, the process of creating the new classes is very easy, since they are built upon already existing classes. The last, but very important advantage is managing the code becomes easy, since the programmer creates several classes in a hierarchical manner, and segregates the code into several modules.

An Example for Inheritance in Java

Here, we take a class A with two variables a and b and a method, `method1()`. Now all these members are needed by another class B, we extend class B from A. We want some additional members in B, for example a variable c and a method, `method2()`. So, these are written in B. Now remember, class B can use all the members of both A and B. This means the variables a,b,c and also the methods method1() and method2() are available to class B. Please observe 'protected' is another *access specifier* in Java that is generally used in inheritance.

```
class A
{
      protected int a;
      protected int b;

      public void method1()
      {
      }
}
class B extends A
{
      private int c;

      public void method2()
      {
      }
}
```

Polymorphism

The word 'Polymorphism' came from two Greek words 'poly' meaning 'many' and 'morphos' meaning 'forms'. Thus, polymorphism represents the ability to assume several different forms. In programming, we can use a single variable to refer to objects of different types and thus, using that variable we can call the methods of the different objects. Thus a method call can perform different tasks depending on the type of the object.

Polymorphism provides flexibility in writing programs in such a way that the programmer uses same method call to perform different operations depending on the requirement.

Example code for Polymorphism in Java:

Let us take a super class 'One' and a sub class 'Two' as shown here:

```
class One
{
      void calculate(int x)
      {
            System.out.println("Square value= "+ (x*x));
      }
}

class Two extends One
```

```
{
    void calculate(int x)
    {
        System.out.println("Cube value= "+ (x*x*x));
    }
}
```

Let us take obj1 and obj2 are objects of classes One and Two respectively. Let us create a reference variable 'ref' for class One as:

```
One ref;
```

If we use this 'ref' to refer to the object of class One, as:

```
ref = obj1;
```

and then call the `calculate()` method, as:

```
ref.calculate(2);
```

It will calculate the square value.

If on the other hand, we use 'ref' to refer to the object of class Two, then 'ref' will call the method of the class Two, as:

```
ref = obj2;
ref.calculate(2);
```

This will calculate the cube value. Observe that the same method call is performing two different tasks.

We should remember that Object oriented approach is a methodology to accomplish tasks in a better way in programming. Languages like C++ and Java which are created based on this methodology are called 'Object oriented programming languages'.

Important Interview Question

What is the difference between object oriented programming languages and object based programming languages?

Object oriented programming languages follow all the features of Object Oriented Programming System (OOPS). Smalltalk, Simula-67, C++, Java are examples for OOPS languages.

Object based programming languages follow all the features of OOPS, except Inheritance. For example, JavaScript and VBScript will come under object based programming languages.

Conclusion

Since the Procedure oriented approach is not suitable for managing bigger and complex projects, another approach called Object oriented approach is invented. Object oriented approach advocates using classes and objects in writing programs. It comes with features like encapsulation, abstraction, inheritance as well as polymorphism. Using these features, if programs are written, it is called Object Oriented Programming System (OOPS). The practical implementation and impact of OOPS in programming will be visited in subsequent chapters.