Introduction to Object Oriented Programming

The term Object Oriented Programming is a relatively new concept in the world of programming languages. Earlier the only style of programming was known as **Sequential or Linear or Procedural Programming**. Every program has two parts: Code and Data. The code part consists of the statements that are executed to perform the job. The data part consists of the variables which hold the necessary values to perform the job. In Sequential programming style more emphasis is placed on the code part and less on the data part. The code statements are executed one by one, sequentially, meaning that you can execute statement number n if you have already executed statement number n-1, and the next statement to execute is statement number n + 1.

In **Object Oriented Programming**, the style is changed. Basically, more emphasis is placed on the data part and emphasis placed on the code part is secondary. Everything you have to consider must be viewed as an **object**. The definition of an object is like the following.

An object is a collection of a set of data and a set of code. The data part is called attributes (formed as variables) and the code part is called functions (formed as executable statements). The relationship between the attributes and the functions is such that if a function is executed, the values of some attribute can be accessed or changed. In other words, to access or change the value of some attribute, we must execute the proper function.

Let us explain with an example. Consider 'chair' to be an object. We can say that any chair in the world must have a set of attributes like height, width, breadth, color, material, weight, price and location. To validate our case, we illustrate three different chairs of different types.

	Chair1	**Chair2**	**Chair3**
Height	3 ft	2.5 ft	3.5 ft
Width	2 ft	2.5 ft	3 ft
Breadth	3 ft	2.25 ft	3.5 ft
Color	Red	Green	Blue
Material	Wood	Plastic	Iron
Weight	5 kg	2 kg	10 kg
Price	500/-	250/-	1000/-
Location	Centre	West	North East

The attribute set described above defines the data part of the three chair objects. You can see that all the three chairs have the identical attribute set. It may happen that the value within the individual attribute could be different for the three objects. For example the three objects have different values for the material attribute, but all of them have the material attribute in their set of attributes.

But as they are objects, the chairs must also have a set of functions (meaning what we can do with these chairs). The set of functions could be like the following:

```
Buy();
Sell();
Repair();
Paint();
Move();
```

As you can understand, if we execute these functions, then one or some of the attribute values will be accessed or changed. For example, to `Buy()` or `Sell()` the chair, its price attribute will be changed. To `Repair()` a chair, we may change its dimensions, weight or material. To `Paint()` a chair its color property will be changed. To `Move()` a chair, we change its location.

The concept of Class

If some objects are found to have a similar set of attributes and a similar set of functions, then these objects can be grouped together into a **class**. A class is a collection of declaration of the attribute variables and the definition of the functions. Once the class has been defined, we can create as many objects of a class as we need just as we declare variables of a data type. In the light of a class, an object can also be defined as an instance or occurrence of a class.

We may declare the CHAIR class as having declared variables like height, width, breadth, color, material, weight, price and location and define the functions like `Buy()`, `Sell()`, `Repair()`, `Paint()` and `Move()`. Now, in order to declare a new object Chair4, all we need to do is write the following statement:

`CHAIR Chair4;` (Equivalent to `int n;`)

And the job is done.

Each object has its own space in memory where it can store its own attribute set, because each object is expected to have a unique attribute set of its own. However, there is no need to store separate copies of the functions for each individual objects, as the coding for the functions need not be changed for each object.

Writing the actual program in an object oriented system is fairly easy. All we have to do is to execute the appropriate function with the appropriate object. For example, if we want to sell Chair2, then we may write:

```
Chair2.Sell();
```

And if we want to move Chair3 then we may write:

```
Chair3.Move();
```

Below we briefly mention some of the important features of the object oriented programming.

Encapsulation

Encapsulation is the technique of putting together the data part (attributes) and the code part (functions) that can operate on the data part into the same single container called a class.

Data Abstraction or Data Hiding

This is the technique of making some part of the class accessible to the outside world while keeping some sensitive part inaccessible. Generally the attributes of a class are defined with a keyword called **private**, which means that these attributes cannot be accessed from outside the class by any means. The functions of the class are generally declared with a keyword called **public**, which means that they can be accessed from outside the class if called using an object of the same class.

Polymorphism

This means the quality of the same thing to exist in different forms. This is an integral feature of object oriented programming that is not found in sequential programming.

Inheritance

This is also another unique feature of object oriented programming. This means the creation of a new class out of an existing class. Inheritance supports the concept of **code reuse** and saves a lot of time and effort to create a new class.

The above notes are submitted by:

Sabyasachi De

MCA

sabyasachide@yahoo.com