UNIT 1 WHAT IS OBJECT-ORIENTED PROGRAMMING?

Structure **Page Nos.**

1.0 INTRODUCTION

When computers were initially introduced, the engineers purely operated them. For a layman, using computers was something like flying an aircraft. So, the genesis of the computer revolution was in a complex machine. The genesis of our programming languages, thus, tends to look like that machine. However, computers have evolved as user-friendlier tools, which are not so much of a machine, but a real world friend that support human expression. As a result, the tools are beginning to look less like machines and more like parts of our minds, and also like other expressive mediums such as writing, painting, sculpture, animation, and filmmaking. Object-oriented programming is part of this movement toward using the computer as an expressive medium.

This unit is an attempt to introduce you what is an object-oriented paradigm and why the industry should move from a procedural paradigm of programming to an object oriented paradigm.

1.1 OBJECTIVES

After going through this Unit, you will be able to:

- describe the term: object, the heart of Object Oriented languages;
- compare and contrast an object oriented language from a procedural language;
- describe the advantages of using object oriented programming, and
- decide when to use procedural language and when to use a object oriented language.

1.2 OBJECT-ORIENTED PROGRAMMING PARADIGM

Whenever we have a problem in hand, we have a very natural tendency to differentiate the Problem Space and Solution Space of the problem, i.e., the place where the problem exists and the place where we try to find out answer. When we use computer to solve a problem, then computer is the “Solution Space”, i.e., the place where you model that problem and the “problem space” is the place where the problem that is being solved exists. All programming languages offer some level of abstraction, and the complexity level for its solution space. A problem that you are able to solve by any Programming language is directly related to the kind and quality of abstraction done by it. The term “Kind” in this context implies:

What is it that you are abstracting?

For example, Assembly language is an abstraction of the machine instruction set. Many higher level languages (such as Fortran, BASIC, and C) were abstractions of assembly language. These languages were improvements over assembly language, but their primary underlying abstraction model still required you to think in terms of the structure of the computer rather than the structure of the problem you are trying to solve. It was the job on the programmer to establish the association between the machine and the Problem by proposing the suitable modules, data structures and algorithms. The effort required to perform this mapping is extrinsic to the programming language. Also, this kind of mapping produces programs that are difficult to write and expensive to maintain.

Thus, the focus of such programming paradigms is on processing, that is, the algorithms were needed to perform the desired computation on structured data. The programming languages support this paradigm by providing functions and facilities for passing arguments to these functions and returning values from functions. In other words, emphasis was to:

Decide the structure of data.

Decide which procedures are required. (What is needed?)

Use the best algorithm available. (How to achieve it?)

For example, A Square root function is:

Given a double-precision floating-point argument, it produces the square root.

```
double sqrt (double arg)
{

//	code for calculation
}
void f() // function does not return a value
{
double root2 = sqrt (2);
//	Get and then print the square root
}
```

Code written within curly brackets express group. They indicate start and end of function bodies. From the point of view of programming, functions are used to create order in a maze of algorithms.

The alternative to modelling the machine is to model the problem you are trying to solve. The object-oriented approach goes a step further by providing tools for the programmer to represent solution entities with respect to the problem space. This representation is general enough that the programmer is not constrained to any particular type of problem. We refer to the entities in the problem space and their representations in the solution space as "*objects*." The idea is that the program should be allowed to adapt itself to the terminology used for the problem. It may appear to be a more flexible and powerful language abstraction than what you have had before. Thus, the idea behind Object-Oriented Programming is to allow you to describe the problem in the terminology of the problem, rather than in terms of the computer where the solution will run. There is still a connection back to the computer, but how? That is what we are going to discuss further.

All the programming languages have traditionally divided the world into two parts- data and operations on data. The data is a static entity and can be changed only by the valid operations. The functions that can operate on data have a finite life cycle of its own and can affect the state of data over their lifetime. Such a division is, of course, on the basis the way computers work. The operations or functions have meaning only when they can act on data or modify it. At some point, all programmers- including object-oriented programmers must lay out the data structures that their programs will use and define the functions that will act on the data.

A procedural programming language like C, may offer various kinds of support for organising data and functions. Functions and data structures are the basic elements of procedural design. But Object-Oriented programming tries to model the design of the program as real world philosophy. It groups operations and data into modular units called objects and lets you combine objects into structured networks to form a complete program. In an object-oriented programming language, objects and object interactions are the basic elements of design. Let us discuss them in more detail.

1.2.1 Object: The Soul of Object-Oriented Programming

Object oriented Programming and Design is all about objects. Traditionally, code and data are apart. For example, in the "C" language, units of code are called functions or operations, while units of data are called structures. Functions and structures are not formally connected in "C". A "C" function can operate on more than one type of structure, and more than one function can operate on the same structure. However, it is not true for object-oriented programs. In Object-Oriented Programming, the data and the operations are merged into a single indivisible unit – an object.

An object has both state (data) and behaviour (operations on data). In that way, objects are not much different from ordinary physical entities. It is easy to see how a mechanical device embodies both state and behaviour. For example, a simple non moving entity: an ordinary bottle combine state (how full the bottle is? Is it open? how much warm its contents are?) with behaviour (the ability to dispense its contents at various flow rates, to be opened or closed to withstand temperatures). It is this resemblance of objects to real things that provides objects much of their power and appeal. They not only can model components of real systems, but also fulfill assigned roles as components in software systems. Therefore, object based programming scheme have advantages, we will discuss them later in this unit.

Objects are the physical and conceptual things we find in the world. Hardware, software, animals, and even concepts are all examples of objects. Everyone's world is built on Objects. For example, for nuclear scientist plutonium, atoms, their speed are all objects. For a civil engineer, bricks, columns, labour are the objects. Finally, for a software engineer developing windows based program, windows, menus, buttons, etc., are the objects for him. An object, has it's well-defined boundary in which it performs its functions while interacting with other objects with its external interface. Objects interact with each other via messages. (Please refer to *Figure1*). The concept of object being an entity can be described as, when we refer to some object in real world we know it will be in some state (in time and place), for example, in nuclear scientist's world the atom will always be in one or the other kind of state (static or moving; and intact or exploded).

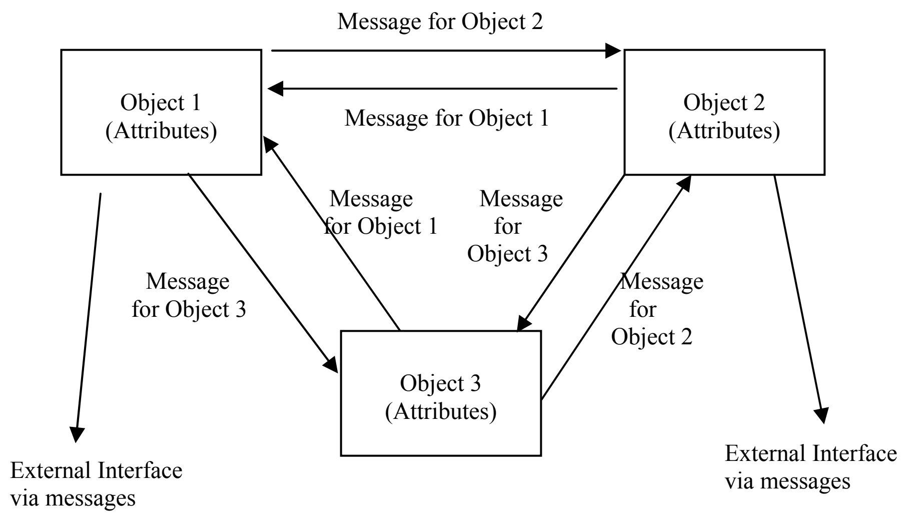


Figure 1: Objects and Interfaces

1.2.2 Object-Oriented Programming Characteristics

The fundamental concept of Object-Oriented Programming is that it allows combination of data and functions/methods/procedures which are working on that data, which did not exist in earlier procedure based programming paradigms.

This fundamental unit is called object. An object has a well-defined interface, which is the only way to access the object's data. The data is thus well organised and hidden. Such hidden data is referred to as encapsulated. Data encapsulation and data hiding are basic and key terms used in OOPS.

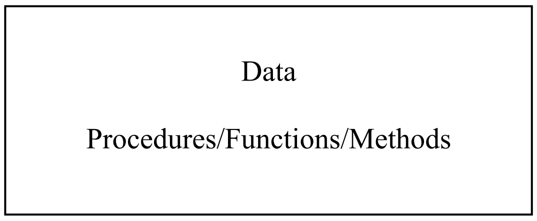


Figure 2: An Object

An object-oriented programming system is composed of multiple objects (see *Figure 3*). When one object needs information from another object, a request is sent asking for specific information, (for example, **a report** object may need to know what is today's date and will send a request to the **date** object).

Figure 3: The object system and messages

These requests are called messages and each object has an interface that manages messages.

A primary rule of object-oriented programming paradigm is that:

"As the user of an object, you should never need to peek inside it."

Why should you not look inside an object?

All communications among the objects is done via messages. Messages define the interface to the object. The object that a message is sent to is called the receiver of the message. Everything an object can do is represented by its message interface. So, you need not know anything about what is in the object in order to use it.

If you look inside the objects, it may tempt you and you would like to tamper with the details of how the object works. Suppose you have changed the object and later the person who programmed and designed the object in the first place decided to change some of these details, then you may be in trouble. Your software may not be working correctly. But as long as you just deal with objects via their messages, the software is guaranteed to work. Thus, it is important that access to an object is provided only through its messages, while keeping the details hidden.

But why should we be concerned about the changes in the object design? Because software engineering experiences have taught us that software do change. A popular saying is that "software is not written, it is re-written." Please remember that some of the costliest mistakes in computer history are because of software that failed when someone tried to change it.

So that is an object. Let us now try to identify the basic characteristics of Object. Oriented Programming?

Five basic characteristics of any object-oriented language representing a pure approach to object-oriented programming are:

1. The basic programming entity is the object. An object can be considered to be a variable that stores data and can perform operation on the stored data itself.

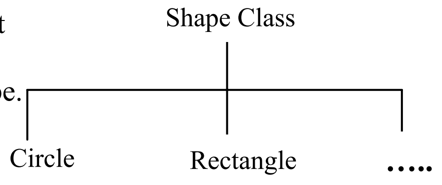
2. An object oriented program is a collection of objects for solving a problem. These objects send messages to each other. A message can be equated to a request to call a function of the receiver object.

3. Each object has its own memory or data that may be made up of other objects. Thus, object-oriented programs are suitable for complex problem solving as they hide the complexity behind the simplicity of objects.

4. Each object can be related to a type, which is its class. An important consideration of a class is that it specifies the message interface that is the messages that can be sent to that type/class of the objects.

5. All object of a particular class can receive the same messages, but may behave differently. This leads to an important conclusion. Let us take an example, a circle object having centre at x=0 and y=0 and a radius of 1 cm. is of the class circle. However, it is also of the type shape. Thus, this object is bound to accept the messages that can be sent to class shape. Similarly, a rectangle object is of type rectangle and also of type shape and will follow messages sent to class shape. Both these objects may be handled using the type shape, but may respond to a message differently on receiving the same message. This is one of the most powerful concepts of an object-oriented programming languages. The concept

Class hierarchy



involves the concepts of inheritance and polymorphism. These concepts are discussed in Unit 2 of this block.

1.3 ADVANTAGES OF OBJECT-ORIENTED PROGRAMMING

The popularity of Object-oriented programming (OOP) was because of its methodology, which allowed breaking complex large software programs to simpler, smaller and manageable components. The costs of building large monolithic software were enormous. Moreover, the fundamental things in object oriented programming are objects which model real world objects. The following are the basic advantages of object-oriented systems:

Modular Design: The software built around OOP are modular, because they are built on objects and we know objects are entity in themselves, whose internal working is hidden from other objects and is decoupled from the rest of the program.

Simple approach: The objects, model real world entities, which results in simple program structure.

Modifiable: Because of its inherent properties of data abstraction and encapsulation (discussed in Unit 2), the internal working of objects is hidden from other objects. Thus, any modification made to them should not affect the rest of the system.

Extensible: The extension to the existing program for its adaptation to new environment can be done by simple adding few new objects or by adding new features in old classes/types.

Flexible: Software built on object-oriented programming can be flexible in adapting to different situations because interaction between objects does not affect the internal working of objects.

Reusable: Objects once made can be reused in more than one program.

Maintainable: Objects are separate entities, which can be maintained separately allowing fixing of bugs or any other change easily.

☞ Check Your Progress 1

1) An object contains data and methods. Even a program written in C has those; then how is the object-oriented programming different from procedural programming?

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

2) How does two-object communicate?

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

3) What are the three most important advantages of OOP?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

1.4 SOME APPLICATIONS OF OBJECT-ORIENTED PROGRAMMING

OOPS has wide following since its inception; it is not only a programming tool, but also a whole modeling paradigm. In addition to general problem solving, two of the upcoming object-oriented applications that are emerging very fast are:

1.4.1 System Software

As an object-oriented operating system, its architecture is organised into frameworks of objects that are hierarchically classified by function and performance. By that, we mean that the whole operating system can be found as made up of objects. The object oriented programming has been a great help for Operating system designers; it allowed them to break the whole operating system into simple and manageable objects. It allowed them to reuse existing codes by putting similar objects in related classes. KDE (a well known desktop of Linux) developers have extensively used the concepts of object oriented programming. Linux Kernel itself is a well known application of object oriented programming.

1.4.2 DBMS

Also known as Object Oriented Database Management System (OODBMS), OODBMS store data together with the appropriate methods for accessing it; the fundamental concept of object oriented programming, i.e., encapsulation is implemented in them which allows complex data types to be stored in database. The Complex datatypes are not supported in Relational Data Base Management Systems. Every data type as well as its relations are represented as objects in OODBMS.

OODBMS have the following features:

- Complex data types can be stored.
- A wide range of data types can be stored in the same database (e.g., multimedia applications).
- Easier to follow objects through time, this allows applications which keeps track of objects which evolve in time.

Applications of OODBMS

The advantages of OODBMS applications are:

- CASE
- CAD
- CAM
- Telecommunications
- Healthcare
- Finance
- Multimedia
- Text/document/quality management

Advantages of OODBMS

- The objects do not require re-assembling from their component tables (in which they are initially stored) each time they are used thereby reducing processing overheads by increasing access speeds.
- Paging is reduced.
- Versioning is easier.
- Navigation through the database is easier and more natural, with objects able to contain pointers to other objects within the database.
- Reuse reduces development costs.
- Concurrency control is simplified by the ability to place a single lock on an entire hierarchy.
- Better data model as based on the ‘real world’ instead of the ‘flattened’ relational model.
- Relationships and constraints on objects can be stored in the server application rather than the client application, therefore, any changes need to be made in one place only. Thus, reducing the need for and risks involved in making multiple changes.

Disadvantages of OODBMS

The disadvantages of OODBMSs are:

- Late binding (discussed in unit 3), which may cause extensive searches through the inheritance hierarchies, may reduce speed of access.
- There are as yet no formal semantics for OODBMS. Relational database can be ‘proved’ correct by means of set theory and relational calculus.
- The simplicity of relational tables is lost.

1.5 THE OBJECT ORIENTATION

Suppose, you want to add two number say, 1 and 2, in an ordinary, non-object-oriented computer language like C. You might write this as:

```
a = 1;
b = 2;
c = a + b;
```

The above code implies that take a number ‘a’, which has the value 1, and number ‘b’, which has the value 2, and add them together using the C language’s built-in addition capability. Take the result, which happens to be 3 in this case, and places it into the variable called ‘c’.

Now, here’s the same thing expressed in C++, which is a pure object-oriented language:

```
a = 1;
b = 2;
c = a + b;
```

You must be wondering that the above code looks exactly the same. You are right, looks the same, but the meaning is dramatically different.

In C++, this says, take the object 'a' which has the value 1, and send it the message "+", which includes the argument 'b' which, in turn, has the value 2. Object 'a', receives this message and performs the action requested to produce a resultant object which in this case has a value 3 and assign this object to object 'c'.

The reason is that objects greatly simplify matters when the data get more complex. Suppose you wanted a data type called list, which is a list of names. In C, list would be defined as a structure.

```
struct list {
<definition of list structure data here>
};

list a, b, c;
a = "Object Oriented";
b = "Programming";
```

Let's try to add these new a and b in the C language:

```
c = a + b;
```

Will it work? No. The C compiler will generate an error when it tries to compile this because it does not understand what to do with addition of two strings. C compilers just understand how to add number, but a and b are not numbers.

One can do the same thing in C++, but this time, list is defined and implemented as a class called a "String".

```
list a, b, c;

a = "Object-Oriented";
b = "Programming";
c = a + b.
```

The first three lines simply create list objects 'a' and 'b' from the given strings. The addition may work if the list class was created with a function/method which specifically "knows" how to handle the message "+". For example, the message plus might simply be used for concatenation of two strings. Thus, the value of C may be "Object-Oriented Programming".

Using Non-Object-Oriented Languages

It is also possible to use objects and messages in non-object-oriented languages. This is done using function calls. Among other things, such an implementation allows sophisticated client-server software to run "transparently" from within ordinary programming languages.

Suppose you want to add a "plus" function to a C program:

```
int plus (int arg1, int arg2)
{return (arg1 + arg2);}
```

This has not really brought you anything yet. But suppose that instead of doing the addition on your own computer, you automatically sent it to a server computer to be performed:

int plus (int arg1, int arg2)

{return server_plus (arg1, arg2);}

The function server_ plus () in turn creates a message containing arg1 and arg2 and sends this message, via a network, to a special object which sits on a server computer. This object executes the "plus" function and sends the result back to you. It is an object-oriented computing via a back-door approach.

☞ Check Your Progress 2

1) State True (T) or False (F)
   (a) The object-oriented programming cannot be used for client server applications True ☐ False ☐
   (b) The kernel of Linux Operating System is implemented using C. True ☐ False ☐
   (c) One of the advantages of object-oriented database management system is that it reduces developmental costs. True ☐ False ☐
   (d) Object Oriented Programming C++ is slower than procedural programming. True ☐ False ☐
   (e) Error handling cannot be done in object-oriented programming. True ☐ False ☐

2) What is the meaning of expression C = a + b in the context of C++, where c, a and b all are complex numbers?

   …………………………………………………………………………………

   …………………………………………………………………………………

   …………………………………………………………………………………

1.6 SUMMARY

The object-oriented programming offers a new and powerful model for writing computer software. The basic backbone being the object, which sends and receives messages, object-oriented programming paradigm speeds up the program development and improves maintainability and modification of a program.

Object-oriented programming requires a major shift in thinking by programmers. The C++ language offers an easier transition via C, but it still requires an object-oriented design approach in order to make proper use of the technology.

Thus, to take full advantage of object-oriented programming, one must program (and think!) using objects.

1.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1) The main difference is in the approach. The data of objects can be modified by the functions of that object/class whereas in procedures/functions, there is concept of local data and global data. The basic advantage of such OOP

scheme, thus, is what operations that can be performed on objects are known and one can easily determined which elements/functions/object have caused an error in data, if any.

2) Through interface messages.

3) 1) Reusability
   2) Maintainability
   3) Modular hierarchical design

Check Your Progress 2

1) a) False
   b) False
   c) True
   d) True
   e) False

2) You as a programmer have to define a meaning for the message '+' as it is not directly defined in the language for complex number objects. How can you do it?

   You will learn about how to do such defining things using C++ in Block 2.