UNIT 1 OBJECT ORIENTED PROGRAMMING

Structure **Page Nos.**

1.0 INTRODUCTION

The computer programs are the means used by human beings for communication with the machines especially the computers. As you all know, programs or the software contain instructions for the hardware to accept inputs, to process those inputs according to the instructions and to produce information as per the instructions contained in the program. The term 'programming' today is used to define the solution to a specific problem to be solved with the help of programs and it is said to define its solutions in terms of its design, creation, testing, debugging, implementation and its maintenance. Throughout the history of programming, ever increasing complexity of the problems to be solved using computers has encouraged the researchers and developers to evolve better means to manage this complexity. Complexity and intended areas of application coupled with some other factors have led to the evolution of a number of programming paradigms. Various programming languages are in use today depending upon various existing programming paradigms.

The structured programming and the Object-Oriented Programming (OOP) paradigms are the two paradigms that have been drawing attention of programmers for last so many years. The term OOP was used by Xerox PARC for the first time in its programming language, Smalltalk referring to the usage of objects as computational units for processing. The language Smalltalk itself got its inspiration from another OOP language called Simula 67 developed under the aegis of Simula Project in late 60s. The feature of inheritance introduced for the first time in Smalltalk allowed it to surpass both Simula 67 as well as other analog programming systems. Simula 67 and Smalltalk paved the way for many other OOP languages including C++ by 1980s.

This unit starts with a discussion on what a program is and what the programming is all about. It further highlights various important programming paradigms focussing basically on the structured and OOP paradigms. Subsequently, you will learn the main concepts involved in the OOP have been presented along with the benefits of OOP.

1.1 OBJECTIVES

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

- understand the concepts of program and programming;

- know about various major computer programming paradigms;
- explain the structured and OOP paradigms and to appreciate the differences between these two;
- gain insight into various concepts that support the OOP; and
- describe the benefits of OOP.

1.2 PROGRAM AND PROGRAMMING

As you might be aware, the two essential components of any computer system are hardware and software. Both hardware and software have their own sets of functionalities which can be interdependent or independent of each other. A computer system is designed to produce the desired results by making the functionalities of both the hardware and the software to converge. The hardware is what we can see, touch and feel e.g. keyboard, mouse, visual display units like monitors, printers etc. Once it has been designed and manufactured to provide a certain set of functionalities, it can not be modified easily. Any modification in the hardware requires lot of effort, time and money. That is why we don't change our hardware very frequently. If the computers are required to carry out only a few predefined operations, these can be very easily embedded in the design of its hardware. But this kind of a computer completely lacks flexibility. In order to provide flexibility to perform some different operation in a computer, most of the existing hardware requires to be replaced with a newer one; whenever a new operation is added or older operations are to be abandoned or modified. Therefore, a computer system always contains a minimum basic hardware which is used by the software to provide lot of flexibility of operations. The software can be modified / replaced with lesser effort, time and money.

As such, a computer is essentially a data processing machine which requires two kinds of inputs for its operations and these are: data and instructions. The hardware of a computer can not produce the desired results unless it is given the requisite instructions and data by the user(s). The data is what needs to be processed by the hardware and the instructions (from within the set of its functionalities) tell this (minimum basic) hardware how to process that data step by step within the realms of set of functionalities so that expected results are achieved. Do you know what is software all about? The software deals with the instructions. The examples of software are Microsoft Office, Microsoft Windows 7, Red Hat Linux, Railways Reservation System, Microsoft Internet Explorer, Google Search Engine etc.

A software is an integrated set of interrelated programs which instruct the hardware as to what to do and how; and is responsible for getting the desired jobs executed by the computer to produce the predetermined outputs.

A program as an independent entity or as part of a software is intended to instruct the hardware to carry out specific task(s) to the satisfaction of the user(s) by providing specific outcomes. So how do you define a program? A program can be defined to be a set of instructions written in a programming language which are given in a fixed sequence to the hardware of a specific computer and executed by its hardware to produce predetermined and expected outcomes. The instructions in a program are written mostly in natural languages (e.g. English, Hindi, French, German, and Chinese etc.) following the syntax (form) and semantics (meaning) of the programming language chosen for writing the program. There are a variety of programming languages available for writing the programs e.g. BASIC, C, C++, Java, Prolog, Lisp, HTML, PHP etc. The sequence of instructions is very important because if the sequence is not correct, the expected results cannot be achieved by the program.

Now, let us see what does programming mean to us? The meaning of the term programming (or computer programming) has been changing rapidly since the idea of a first program was envisaged. Initially the computers were used to solve the mathematical problems with the help of calculations. Hartee in 1950 suggested that

the process of preparing a calculation for a machine can be broken down into two parts, 'programming' and 'coding'. He described programming as the process of drawing up the schedule of the sequence of individual operations required to carry out the calculation. Before the availability of assemblers, coding was in fact, a very tedious and time consuming task. Soon, programming became the major activity in this process.

In 1958, Booth proposed that the process of organizing a calculation can be divided into two parts, a) the mathematical formulation, and b) the actual programming. With the passage of time, the definition of programming has kept on evolving and at present programming is considered to be the process of writing programs and may include activities as diverse as designing, writing, testing, debugging and maintaining the code of a program. In normal conversation, programming is described as the process of instructing the computer to do something desired and useful for the user with the help of a programming language.

☞ Check Your Progress 1

Objective type Questions:

1) How are hardware and software related in a general purpose computer?
   a) They are independent of each other
   b) Hardware has to be dependent on software
   c) Software (System) has to be dependent on hardware
   d) None of these

2) Which is of the following is not a part of the definition of the program?
   a) Instructions
   b) Sequence
   c) Data
   d) Desired output

3) Which one of the following is not a programming language?
   a) C++
   b) HTML
   c) BASIC
   d) English

Answers to short type questions:

1) Why can hardware of a computer not produce the desired results in the absence of instructions (program / software)?

   ……

   ……

   ……

2) How is a program related to software?

   ……

   ……

   ……

3) Why does hardware not provide flexibility of operations to the user(s)?

   ……

   ……

   ……

1.3 PROGRAMMING LANGUAGES

You must appreciate the fact that the programming languages are created by, we, human beings. These languages are used to communicate instructions to the machines especially computers so that the programs can control the behaviour of the hardware of the machines to get desired results. Basically, the hardware of the computers understands only the language of the hardware which is called the machine language. The hardware is unable to understand and decipher any program written in any other programming language. Moreover, every type of a CPU has its own machine language. Therefore, in order to make the hardware of a computer understand the instructions contained in a program written in any other programming language, a mechanism called 'translator' is required. This translator converts the program written in programming languages other than the native machine language of the CPU (hardware) into the native machine language of a particular CPU on which this program is intended to be executed. Every programming language must have its own translator for the programs written in it to be executed or run on the computer hardware. Various types of translators available can be categorized into assemblers, interpreters or compilers.

The primitive or the first generation of programming languages were called machine languages and the symbols like '0' and '1' were used to write programs under this category of programming languages. The second generation of programming languages were called the assembly languages and mainly used mnemonics to construct a program. Both of these generations of programming languages were CPU dependent i.e., every type of a CPU will have its own machine and assembly language. The third generation of programming languages was called high level languages as these programming languages were independent of the CPU of the hardware being used and the instructions written in the programs were just like the instructions given in natural languages. The third generation languages are known as 3GL languages. The current generation of the programming languages are called the fourth generation languages or the 4GLs. These languages represent the class of programming languages that are closest to the human (natural) languages.

Based on the intended use of domain of use, the programming languages are broadly classified as imperative programming languages where imperative sentences are used in a program to issue commands in terms of instructions; and declarative programming languages where declarative instructions are used in a program to assert the desired result. But a more common paradigm classifies these languages into imperative, functional, logic programming and object-oriented languages. Table 1.1 presents the summary of main features of these programming paradigms.

Table 1.1: Summary of Main Features of Programming Paradigms

Paradigm	**Key Concepts**	**Program**	**Program Execution**	**Result**
Imperative	Command (instruction)	Sequence of commands	Execution of commands	Final state of computer memory
Functional	Function	Collection of functions	Evaluation of functions	Value of the main function
Logic	Predicate	Logic formulas: axioms & a theorem	Logic proving of the theorem	Failure or Success of proving
Object-oriented	Object	Collection of classes of objects	Exchange of messages between the objects	Final state of the objects

Table 1.2 shows some of the examples of programming Paradigms.

Table 1.2: Programming Languages under Programming Paradigms

Paradigm	Example
Imperative	Algol, Pascal, C, Ada
Functional	Lisp, Refal, Planner, Scheme
Logic	Prolog
Object-oriented	Smalltalk, Eiffel, C++, Java

There exist certain programming languages that inherit the features of more than one paradigms and the examples of some modern programming languages are presented in Table 1.3.

Table 1.3: Programming Languages under Two Programming Paradigms

Paradigms	Example
Imperative + Object-oriented	Object Pascal, C++, Java, Ada-95
Functional + Object-oriented	Clos
Logic + Object-oriented	Object Prolog

1.4 STRUCTURED PROGRAMMING PARADIGM

The structured programming paradigm is a sub discipline of procedural programming under the category of imperative programming paradigm. Most of the present day procedural programming languages include the features that encourage structured programming. Do you know who proposed this paradigm in the first instance? It was first proposed by two mathematicians Corrado Bohm and Guiseppe Jacopini who proposed and demonstrated that a computer program may contain just three structures namely decisions, sequences, and loops.

Structured paradigm is based on the principle of building a program from logical structures.

Any program can be created by breaking large programs into smaller modular routines and imposing these logical structures. That is why structured programming is also sometimes known to follow the concepts of modular programming. This paradigm generally follows the top down approach where the complex programming blocks are broken down into smaller blocks maintaining a well defined structure and organization of the overall program. It discourages the use of global variables and instead encourages to use variables that are local to each of the blocks. Moreover, the use of GOTO statement is completely forbidden in the languages supporting this paradigm. Some of the languages that follow this paradigm are Pascal, C, C++, Java, Ada etc. in contrast to non-structured programming languages like BASIC, COBOL, FORTRAN etc.

The structured programming follows the principle of divide and conquer. A solution of a problem can be said to consist of (or include) a set of tasks, on the same lines, a program can also be designed to perform a set of tasks by dividing it into the task performing blocks.

Under unstructured programming paradigm, a) mostly, all the program code is written in a single continuous main program, b) logic is difficult to follow within the program,

c) code from other programs is hard to incorporate, d) it is difficult to test specific portions of a program, and e) program is difficult to debug and maintain. In contrast, structured programming is defined as a programming paradigm which follows certain set of quality standards to create programs that are more reliable and readable; and easier to maintain. Under this paradigm, the aim is that before the code is written, the structure of a program is required to be defined clearly and a decision to attach other programs and libraries be taken.

Some of the major advantages and disadvantages of structured programming are given below:

1.4.1 Advantages of Structured Programming

a) Complexity can be reduced using the concepts of divide and conquer.

b) Logical structures ensure clear flow of control.

c) Increase in productivity by allowing multiple programmers to work on different parts of the project independently at the same time.

d) Modules can be re-used many times, thus it saves time, reduces complexity, and increases reliability.

e) Easier to update/fix the program by replacing individual modules rather than larger amounts of code.

f) Ability to either eliminate or at least reduce the necessity of employing GOTO statement.

1.4.2 Disadvantages of Structured Programming

a) Since GOTO statement is not used, the structure of the program needs to be planned meticulously.

b) Lack of encapsulation.

c) Same code repetition.

d) Lack of information hiding.

e) Change of even a single data structure in a program necessitates changes at many places throughout it, and hence, the changes become very difficult to track even in a reasonably sized program.

f) Not much reusability of code.

g) Can support the software development projects easily up to a certain level of complexity. If complexity of the project goes beyond a limit, it becomes difficult to manage.

1.5 OBJECT-ORIENTED PROGRAMMING PARADIGM

Simula was the first programming language developed in the mid-1960s to support the object-oriented programming paradigm followed by Smalltalk in the mid-1970s that is known to be the first ‘pure’ object-oriented language. Eiffel, Java, C++, Object Pascal, Visual Basic, C# etc are the other OOP languages that came into existence later on, all having different complexities of syntax and dynamic semantics.

The main motive of the developers of programming languages over the years has always been to create such programming languages that are close to human (i.e. natural) languages. The way we perceive and interact with the things in our day-to-day lives, the representation of programming constructs should closely match the same. Hence came into existence the concept of 'objects' and 'object-oriented programming paradigm'.

The real world can be considered to be made up of various objects with which the human beings regularly interact e.g. a ball pen, a tooth brush, a vehicle, a foot bear or a house etc.

Every object has certain defining properties which distinguish it not only from different types of other objects but from the similar types of objects too. If we take an example of an object like a ball pen, its defining property may be the colour in which it can write, length, shape, unique manufacturing code etc. Some of these properties not only distinguish the ball pen object from the tooth brush object but also distinguish individual ball pens objects. It must be understood clearly that no two objects in the physical world, even of same type, are identical since no two objects can have the value of all its defining properties same.

Likewise, every object has certain functions associated with it and all similar types of objects are supposed to support these. Although, Some of these functions can be the same as associated with different types of objects. In case of a ball pen, one of the functions associated with each type of ball pen object, is to write and another associated function is the provision to hold it in hands conveniently. All the ball pen objects support both these functions. Incidentally, all the tooth brush objects also support the function of holding them in the hands conveniently but, in addition, support other functions like brushing the teeth too.

This concept of objects borrowed from the real world has been the basis of the object-oriented programming (OOP) paradigm and this paradigm is a direct consequence of an effort to have a programming language closely matching the human behaviour. This OOP paradigm is all about creating program(s) dealing with objects where these objects interact with one another to achieve the overall objectives of the program. Every object in the programs has certain defining properties called attributes (or instance variables) possessing supporting values for each of the attributes and some associated functions (normally called methods or operations). As in the real world objects, no two objects in a program can have the same values of all the attributes.

At times, instead of dealing with individual objects, it is convenient to talk collectively about a group of similar objects where all the objects of this group will have the same set of attributes and methods. In the object-oriented programming paradigm parlance, this collection of objects corresponding to a particular group is known as a class. All programs under this paradigm contain a description of the structure (corresponding to attributes) and behaviour (corresponding to methods) of so called classes. In a program, various objects are created from these classes. The process of creation of an object from a class is called 'instantiation' and the object created is known as an instance of the class. Every object created will have a 'state' associated with the description of the structure in the class from which it has been instantiated. The state of an object is defined by the set of values assigned to its corresponding attributes (and stored in the memory) of the object.

Therefore, we can say that a class is used to represent a set of objects having same structure and behaviour. So, how do we define a class? A class is defined to be a template or a prototype so that a collection of attributes and methods can be described within it and this definition can be used for creating different objects within a program. It is this concept of encapsulating the data and methods within the objects that provides the programmers with flexibility within the OOP paradigm because an object can be extended or modified without making changes to its external interface or other classes/objects in the program. Various classes may exhibit features like inheritance and polymorphism of methods.

When a program is executed, various objects are created with their corresponding states and these objects interact with one another by exchanging messages, the messages thereby causing the modification of their states. Modification in the state of an object is said to have occurred when the values of one or more of its attributes (instance variables) change due to the interaction. All the objects created are made to exhibit a behaviour through their corresponding methods such that the program produces the desired results once its execution is over. A program in this paradigm therefore, becomes a collection of cooperating and interacting objects instead of just a list of objects.

For the sake of an example, a Maruti Wagon-R could be an object. It would have a state depend upon whether its engine is running or not. Also it would have a behaviour, like starting ignition of its engine or stopping the ignition of its engine and this behaviour is responsible for changing its state from 'engine is not running' to 'engine is running' or from 'engine is running' to 'engine is not running'.

In a different example, we can take object ABC representing a student having a state defined by its attribute named RESULT. A part of the behaviour of this object could be reflected through 'compute result'. If this student has not appeared in the examination yet, the state of this object defined by the attribute RESULT may be NOT DECLARED. But once this student has appeared in the examination and the marks obtained by this student are available, the behaviour of this object changes the state of the object ABC by using the method 'compute result' from NOT DECLARED to either FAIL or PASS with percentage of marks.

1.6 STRUCTURED Vs OBJECT-ORIENTED PROGRAMMING

The OOP, can be considered to be a type of structured programming which uses structured programming techniques for program flow, but adds more structure for data to be modelled. We have highlighted some of the basic differences between the two as under:

1) OOP is closer to the phenomena in real world due to the use of objects whereas structured programming is a bit away from the natural way of thinking of human beings.
2) Structured programming is a subset of object-oriented programming. Therefore, OOP can help in developing much larger and complex programs than structured programming.
3) Under structured programming, the focus of a program is on procedures or functions (behaviour) and the data is considered separately (data structures are not well organized within the program) whereas in OOP, data (structure) and methods (behaviour) both are in collective focus.
4) In OOP, the basic units of manipulation are the objects whereas in case of structured programming, functions or procedures are the basic units of manipulation.
5) The focus of a program is on manipulation of data in structured programming whereas the focus of OOP is on both data (structures and states of objects) as well as on its manipulation (behaviour of objects).
6) In OOP, data is hidden within the objects and its manipulation can be strictly controlled whereas in structure programming the data in the form of variables is exposed to unintended manipulation too.

7) The OOP promotes the reuse of classes and their parts of the code too. In addition, it also supports the inheritance of state and behaviour. This feature is missing in structured programming.
8) The OOP supports polymorphism of operations.
9) The concepts of OOP have got integrated with most of the prominent object-oriented methodologies of software development in a better way and help in reducing the development effort and time as compared with that of structured methodologies based on structured programming.
10) The structured programming normally emphasized on single exit point in their constituent functions or procedures. Since each function/procedure allocates some memory to itself for storing the values of its variables and code, before the exit, there must be a provision for deallocating this memory. Otherwise, more and more of the memory gets occupied by each function/procedure and in large programs, there may occur a shortage of memory for use by other functions/procedures and the processing can get slower or even completely halted. When in any function/procedure, memory is allocated, but not deallocated, a memory leak is said to have occurred (the memory has leaked out of the computer) in it.

1.7 OBJECT-ORIENTED PROGRAMMING CONCEPTS

The Object-Oriented Programming is based on sound principles and provides the developers of various object-oriented programming languages with a variety of new concepts to be incorporated in those languages. Some of the commonly found important concepts in most of the OOP languages are as given below:

Abstraction (data)

Abstraction is a technique which allows the hiding or elimination of the irrelevant; and focussing on the essential. According to IEEE 1983 definition, abstraction is defined as a view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information. There are different levels of abstraction. As in real life, for an example of a car, a buyer would see the car at a different level of abstraction, designer has different level of abstraction to look at it, a mechanic sees it at another level of abstraction, a junk yard owner sees the car at an altogether different level of abstraction. The buyer is interested in the colour, mileage, shape, manufacturer etc; the designers are concerned about the minute details like designing the fuel tank, ignition system, electrical parts and their wiring, breaking system etc; a mechanic may be concerned about how to test the battery, how to open and reconnect various parts etc, spare parts etc; and the junk yard owner is interested only in how much reusable metallic and plastic parts are there in the car. If we, therefore, apply the same concept of abstraction to a program in a given context, a programmer hides or eliminates the irrelevant attributes and methods and use only the attributes and methods that are relevant for a given class or an object.

If we take another example of a class ‘student’, the class may have plenty of attributes like, name, roll number, father’s name, mother’s name, date of birth, class, year or semester, course, marks obtained, address for correspondence, height, weight, colour of hair, colour of eyes, size of the shoes they wear, no. of teeth, finger prints, IQ level etc. The list can be very long. But, when we use this ‘student’ class in some program, not many of these attribute would be required to be used. Only those attributes which are of interest (i.e. the attributes that are required to define the state of the program at any point of time during the execution of the program) shall be included in the definition of the class and rest of these attributes shall never be used and hence be not

included in the program. Same is true for behaviour depicted by the methods of the class 'student'.

Information Hiding

Information or data hiding in an object is characterised by its knowledge of a design decision which it hides from all other objects. The interface of an object is chosen to reveal only the desired data or working of the object. According to the definition of information/data hiding given by Booch in 1991, it is the process of hiding all details of an object that do not contribute to its essential characteristics; typically, the structure of an object as well as the implementation of its methods is hidden from other objects. There can be two types of information hiding: functional information hiding related to the hiding of implementation details of methods (behavioural information of a particular object) and data hiding (structural information of a particular object). As in the case of abstraction, there are varying degrees of information hiding. Some languages like C++ also allow varying degree of visibility of objects like public, private and protected. So the mechanism of information hiding is said to provide a strictly controlled access to the information enclosed within the capsule.

Encapsulation

In 1991, Rumbaugh and others defined encapsulation as consisting of separating the external aspects of an object which are accessible to other objects, from the internal implementation details of the object, which are hidden from other objects. According to Booch, it is also known as information hiding and it prevents clients from seeing the object's inside view, where the behaviour of the abstraction is implemented. In reference to classes and objects in OOP, encapsulation is the process of enclosing within these classes and objects the attributes and the methods. It is the programmers who specify what information in an object can be shared with other objects. Figure 1.1 illustrates the concepts of encapsulation and information hiding.

Encapsulation refers to how the implementation details of a particular class are hidden from all objects outside of the class.

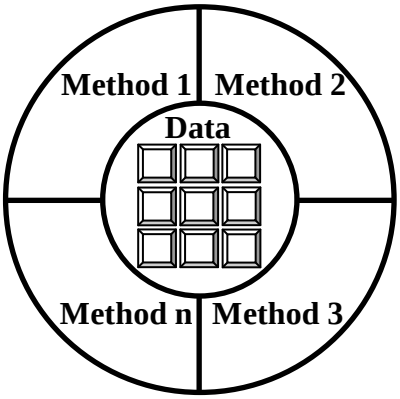

Figure 1.1: Encapsulation & Information Hiding Concept

Do you think information hiding and encapsulation mean the same thing? No, information hiding cannot be treated as encapsulation, both are related but altogether different aspects e.g. an array or a record structure also encloses the information but this information cannot be said to be hidden. It is true that the encapsulation mechanism like classes and objects hide information but these also provide visibility of some of their information through well defined interfaces.

Classes and Objects

A class is a collection of similar entities which have same structure and exhibit same behaviour. These are used to describe something in the real world like places,

organizations, roles, things, occurrences etc. A class is said to describe the structure and behaviour of these sets of similar entities called objects. As opposed to actual objects, the class gives a general description of these objects like a template, blueprint or a pattern; and contains the definitions of all the attributes and methods which will become the part of each object created from the class. Only after a class has been defined, specific instances of the class can be created and these instances are called the instances of that class. The process of creation of these instances as objects of the class is called instantiation. Table 1.4 cites certain examples of classes and their objects for your ready reference.

Table 1.4: Examples of Classes & Corresponding Objects

Type	Example of Class	Example of Object
Place	Hill station	Shimla
Organization	University Department	Computer Science
Occurrence	Alarm	Fire alarm
Role	Teacher	Manoj Kumar
Thing	Car	Maruti Wagon R

Figure 1.2 shows an example of a class having an identifier as 'Teacher', its structure defined by attributes 'Name' of type 'String' and 'Age' of type 'Integer' depending upon the particular context. The behavior of this class in a using abstraction is depicted by a single method 'evaluate'. In this particular context, the programmer is required to focus only on name and age of the teachers as part of the structure of this class and in the evaluation method as part of behavior of this class.

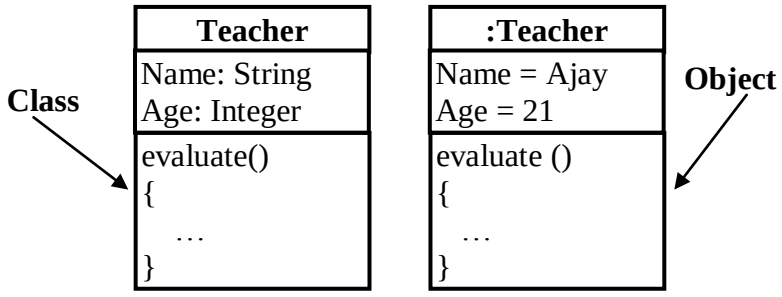

Figure 1.2: Concept of a Class and a Corresponding Object

In different contexts, there may be different sets of attributes and methods of interest for the programmer. An object (instance) created from this class is also shown in this figure having values of attributes 'Name' and 'Age', these values of attributes at any point of time also describe the state of this object. The behaviour of an object is defined by the set of methods which can be applied on it.

Message Passing

In object oriented languages, you can consider a running program under execution as a pool of objects where objects are created for 'interaction' and later destroyed when their job is over. This interaction is based on 'messages' which are sent from one object to another asking the recipient object to apply one of its own methods on itself and hence, forcing a change in its state. The objects are made to communicate or interact with each other with the help of a mechanism called message passing. The methods of any object may communicate with each other by sending and receiving messages in order to change the state of the object. An Object may communicate with other objects by sending and receiving messages to and from their methods in order to

change either its own state or the state of other objects taking part in this communication or that of both. An object can both send and receive messages.

The messages are sent and received by passing various variables among specific methods using the signatures (a term that is not prevalent in common parlance) of the methods. Every method has a well defined and structured signature. The signature of a method is composed of: a) a type, associated with the variable whose value after execution of the method, would be returned to the object that would invoke the method; b) the types of a specific number of variables and the order associated with these variables whose values would be passed to the method before execution of the method starts. All these variables have a well defined format and corresponding values at any instant of time available for communication during the execution of the program. Figure 1.3 shows an example of a signature for a method 'evaluate'.

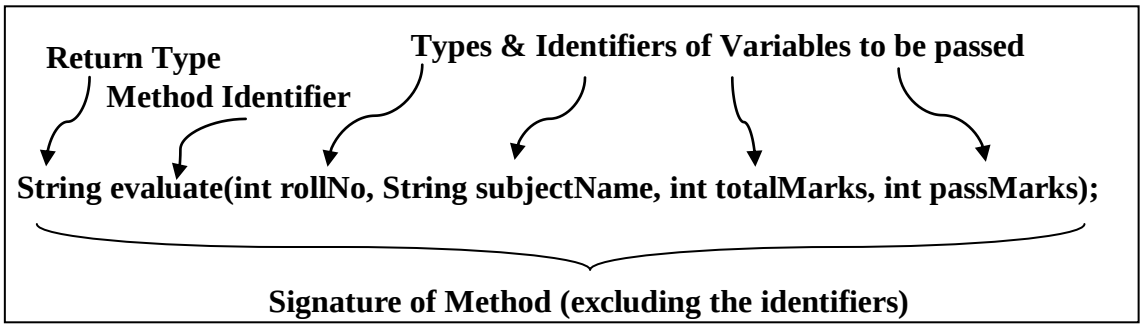

Figure 1.3: Signature of a Method

Interface

Every class defines an interface for itself and its objects use only this interface for all types of communications. We may say that an interface of a class or an object is the collection of signatures of all the methods contained in the class or the object. It is through this interface, the objects communicate with themselves or with other objects by passing value of variables to and fro and hence, in the process, changing their own state or hat of other objects or that of both. As the signatures of various methods of an object are well structured and precisely defined, therefore, the interface of an object also has a well defined precise structure. As you can see, figure 1.4 shows the mechanism of message passing between two objects through their interfaces. Various OOP languages have different mechanisms to implement the interfaces.

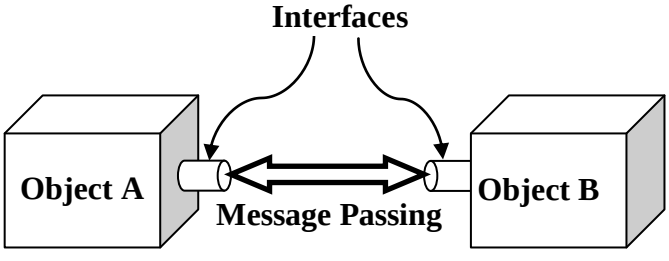

Figure 1.4: Objects Interacting through Interfaces

Association

The classes and hence the corresponding objects in OOP languages are in relationship with one another. Various kinds of vital relationships are association, aggregation, inheritance etc. An association is the term used to represent the relationships among various objects of one or more classes. An illustration of association is given in figure

1.5. The association here has been represented by a simple straight line whereas the classes have been represented by rectangles.

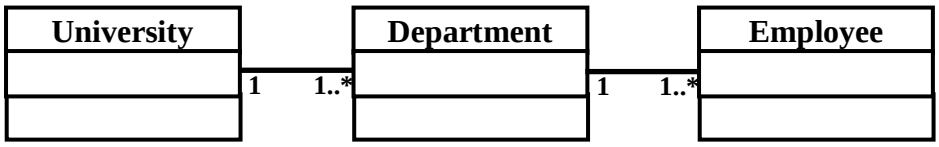

Figure 1.5: Association & Multiplicity among Classes

The term ‘multiplicity’ represented by a numerical specification, is used to indicate how many objects of one side of an association are connected with how many objects on the other side. Common categories of multiplicities have been enlisted in Table 1.5.

Table 1.5: Various Categories of Multiplicity

0..1	No instance, or at the most one instance
1	Exactly one instance
0..*, *	Zero or more instances
1..*	One or more instances

Aggregation is a special form of association. It is the composition of an object out of a set of its parts. A university, for example, is an aggregation of departments, employees, students, class rooms, faculty rooms, laboratories and so on. Some of these parts may further be the aggregations of some other parts. Sometimes, aggregation is also known as a ‘whole-part’ hierarchy (university is ‘whole’ and department is a ‘part’) and represented as ‘has-a’ relationship (a university has-a department).

Inheritance

Can you guess what inheritance is? Inheritance is a mechanism by virtue of which the classes inherit the attributes and methods of some other class(s). In a sense, we can say, inheritance is a way to reuse the code of some existing or already defined classes. The classes, in this case, are said to have ‘a-kind-of’ and ‘is-a’ relationship with the other classes. Using this property of OOP, one class can extend other classes by including additional methods and/or attributes (variable). The original class is called the ‘superclass’ of the extending class and the extended class is called the ‘subclass’ of the class that is being extended. The subclass is sometimes known with the name of ‘derived’ class and the superclass with the name of ‘base’ class. The derived or subclasses classes can further be used to have their own derived subclasses. This kind of a relationship of classes through inheritance gives rise to an inheritance hierarchy of the classes. Can you explain, why?

As an example, if you take ‘car’ as a superclass; then you can treat SUV, sedan, sports car, roadster as its subclasses. All or some of these subclass cars have some common attributes (structure) and methods (behaviour). But the structure and behaviour of these subclass cars is not restricted to those of the superclass ‘car’. A subclass car can contain methods and attributes in addition to those inherited from the superclass ‘car’ e.g. a sports car will generally have only two doors whereas a sedan will have four. The subclasses can also contain methods that override the methods they have inherited to have unique implementation of these methods. Another example of inheritance can be that of a ‘geometric figure’ as a superclass and a ‘circle’ and a ‘triangle’ as its subclasses as shown in figure 1.6.

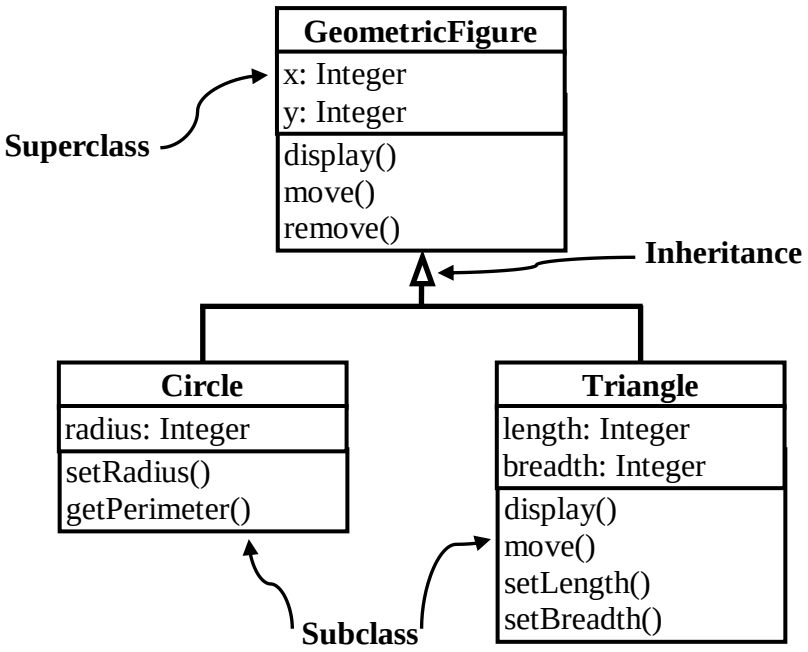

Figure 1.6: Inheritance among Classes

Various categories of Inheritance which are often used in OOP are single level, multi-level, multiple inheritance etc.

Polymorphism

Polymorphism or the ability to appear in many forms, is one of the vital primary characteristic concepts of OOP. 'Poly' means 'many' and 'morph' means 'form'. In reference to OOP, it is an ability of assigning different meanings to entities such as variable, methods or objects so that these can be made to exhibit more than one form. It provides the programmers with the flexibility of processing any object differently depending upon their data types. Using this concept, a programmer can redefine various methods of the classes derived from their base classes. Objects of different types can receive the same message and respond in different ways provided these objects have the same method definition (i.e. interface). The calling object, also sometimes known as the client, need not know what type of object it is calling, the only thing that it needs to know or ensure is that the called object has a method of a specific name with defined arguments. Polymorphism is more often than not applied to derived classes, where the methods of the parent class are replaced with those having different behaviours. It is the concepts like inheritance and polymorphism that together make OOP flexible and easy to extend.

Do you know how many categories of polymorphisms exist? There are two categories of polymorphisms; static or compile-time and dynamic or run-time. In static polymorphism, which form of the method (from among the various available forms) is to be called and executed is decided during the time of compilation, the example being 'Method Overloading'. In method overloading, same method name having different parameters is used more than once in the same class. Which method is to be called and executed depends upon the parameters passed by the calling method and is decided during the compilation of the program. The dynamic polymorphism is applied in the form of method overriding which means there can exist two or more methods in a program which have the same signature (name; return type; type, number and order

of arguments to be passed) having different implementations. In figure 1.7 these two types of polymorphisms are explained.

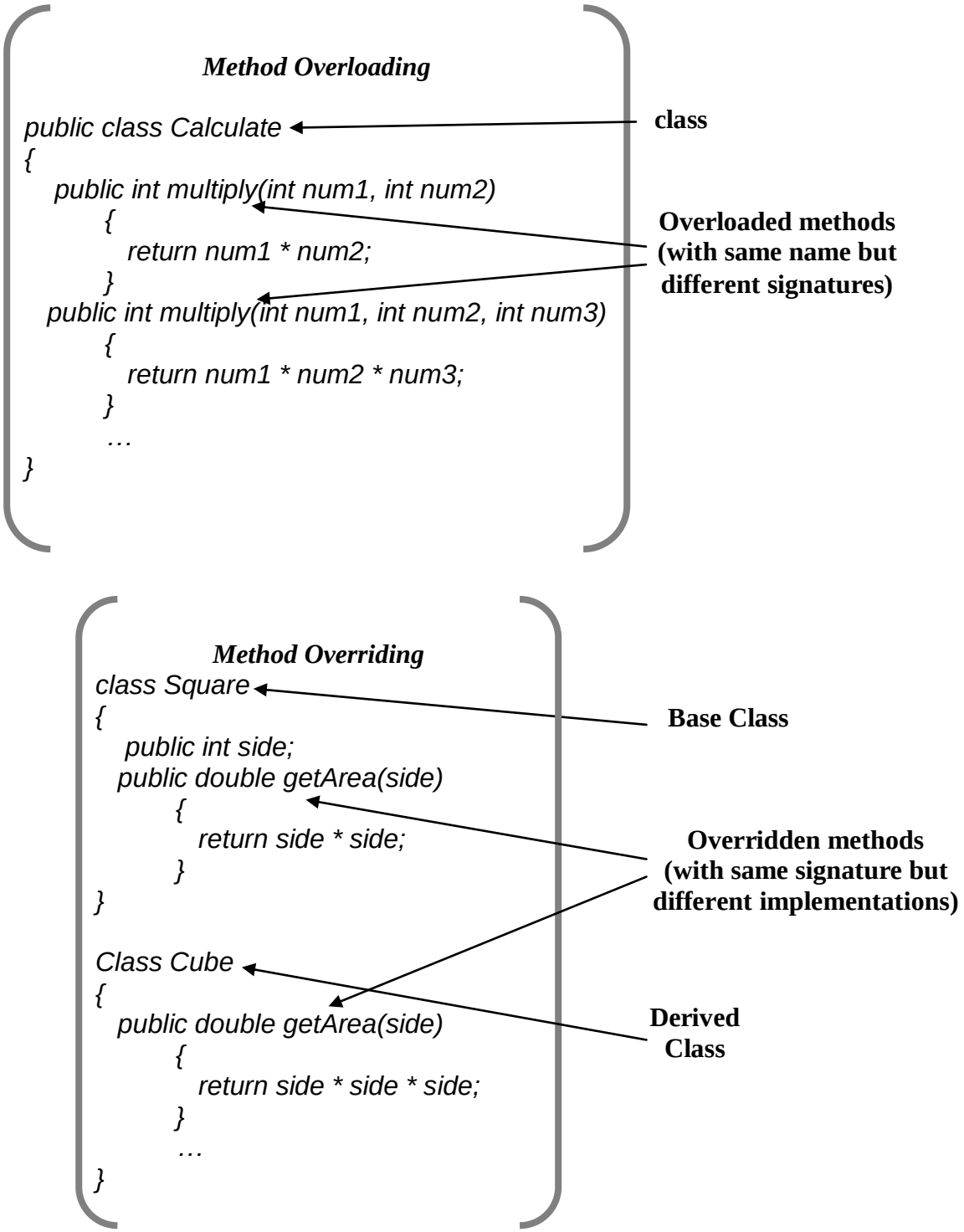

Figure 1.7: Two Different Types of Polymorphism

In addition to object-oriented programming, the programmers sometimes also use object-based programming languages. So, how is object-based programming different from object-oriented programming? According to Rumbaugh, object-oriented programming should be supported by the languages which have at least the following four features:

a) Identity which means the quantization of data in terms of entities called the objects that are discrete and distinguishable;
b) Classification into classes i.e. grouping of objects of same structure and behaviour;
c) Polymorphism i.e. depiction of different behaviour of same operations on different classes; and
d) Inheritance in terms of sharing of structure and behaviour among classes in a hierarchical relationship.

But, what about the languages that support similar kinds of features? As such, there are certain languages that may support some of these feature but not all. The languages that support only some of these features like identity and classification (and may be polymorphism too) do not qualify to be called object-oriented programming languages. These programming languages are called object-based programming languages. Visual Basic is one such programming language which supports objects and classes but not inheritance and that is why it is called an object-based programming language. In contrast, VB.NET is an object-oriented programming language. Fortran 90 is another example of object-based programming language that does not support inheritance. Another example is JavaScript, a language that does not have classes. In this language, the objects can be made to inherit code and data directly from the template objects.

1.8 BENEFITS OF OOP

By now, you might have understood the basic concepts of object-oriented programming. Therefore, you are in a better position to appreciate the following as some of the major benefits of OOP:

1) As OOP is closer to the real world phenomena, hence, it is easier to map real world problems onto a solution in OOP.
2) The objects in OOP have the state and behaviour that is similar to the real world objects.
3) It is more suitable for large projects.
4) The projects executed using OOP techniques are more reliable.
5) It provides the bases for increased testability (automated testing) and hence higher quality.
6) Abstraction techniques are used to hide the unnecessary details and focus is only on the relevant part of the problem and solution.
7) Encapsulation helps in concentrating the structure as well as the behaviour of various objects in OOP in a single enclosure.
8) The enclosure is also used to hide the information and to allow strictly controlled access to the structure as well as the behaviour of the objects.
9) OOP divides the problems into collection of objects to provide services for solving a particular problem.
10) Object oriented systems are easier to upgrade/modify.
11) The concepts like inheritance and polymorphism provide the extensibility of the OOP languages.
12) The concepts of OOP also enhance the reusability of the code written.
13) Software complexity can be better managed.
14) The use of the concept of message passing for communication among the objects makes the interface description with external system much simpler.
15) The maintainability of the programs or the software is increased manifold. If designed correctly, any tier of the application can be replaced by another provided

the replaced tier implements the correct interface(s). The application will still work properly.

☞Check Your Progress 2

Objective type Questions:

1) Which of the following is not a disadvantage of structured programming?
   a) Availability of information hiding b) Lack of encapsulation
   e) Non-availability of GOTO statement d) None of these

2) Which is of the following is not a concept associated with OOP?
   b) Information hiding b) Clauses
   d) Abstraction d) Message Passing

3) Which one of the following describe method overloading the best?
   a) Same signature, different implementation
   b) Same name, different signatures
   c) Different name, same signatures
   d) Different name, different signatures

Short Answer type Questions:

1) What is the signature of a method?

   ……

   ……

   ……

2) Differentiate between information hiding and encapsulation.

   ……

   ……

   ……

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

   ……

   ……

   ……

1.9 SUMMARY

The programs are the means through which we can make the computers produce the useful desired outputs. Out of a variety of programming paradigms being used by practitioners as well as the researchers, the structured and the object-oriented programming paradigm and corresponding structured and object-oriented programming have been in focus for quite some time now. In this unit, you studied that the structured programming languages initially helped in coping with the inherent complexity of the softwares of those times but later on, were found wanting in the handing the same as far as the software of present days are concerned. In the backdrop

of this, you also studied the advantages and the disadvantages of the structured programming.

Next, you were introduced to the concepts of object-oriented programming paradigm and it was illustrated as to how this paradigm is closer to natural human thinking. Subsequently, an overview of the differences between the structured and object-oriented programming paradigms was presented to you so that you can clearly understand these intricate differences. After that, the basic concepts of object-oriented programming well supported by relevant illustrations were introduced to you. Here we first defined abstraction, encapsulation and information hiding elucidating the difference between the last two. It was followed by the illustrations of some more concepts of object-oriented programming like classes, objects, message passing, interface, associations, inheritance and polymorphism. In the end, you saw what the various benefits of OOPs are and how can these help in producing good quality software.

1.10 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

Answers to Objective type Questions:

1) c 2) c 3) d

Answers to Short Answer type Questions:

1) The hardware of a computer does not do anything on its own unless it is provided with instructions in the form of a program or software. These instructions are decoded and executed by the hardware to produce the desired result after getting the instructions from the user for doing so. Therefore, if no instructions are given to the hardware by the user, the hardware will not be able to do anything and hence the desired result will not be produced.
2) Software is a set of related programs which provide specific instructions to the hardware so that the intended results are achieved when the software is used by the computer hardware. A software may consist of a number of programs that are related to each other in such a way that these programs shall be interacting with each other while in execution. It is this interaction among various related programs which helps users to get their intended goals achieved through the execution of a particular software.
3) To provide any functionality through operations to the users, various hardware components are needed to be interconnected based upon the circuit diagram (design) and their operations need to be strictly controlled and synchronized especially with reference to time. These components along with the defined interconnections are then hardwired. Once the components are hardwired, they can not be changed. Even if a small change in functionality through operations is required to be carried out, a new circuit diagram needs to be designed and a new set of components needs to be interconnected all over again. This process is very expensive, wasteful and time consuming. In contrast, the software can be changed any numbers of times without much hassles.

Check Your Progress 2

Answers to Objective type Questions:

1) a 2) c 3) b

Answers to Short Answer type Questions:

1) The signature of a method consists of:
   a) Return type of the method
   b) Number of arguments to be passed
   c) Types of each of these arguments
   d) The sequence of these arguments

   It is through the signature of a method, the mechanism of message passing is supported for interaction within an object or among various objects of a program. Whenever any method of an object intends to communicate with another method of the same object or some other object, the message to be sent from the transmitting object will have to adhere to the format of the signature of the receiving method.

2) The mechanism of information hiding is said to provide a strictly controlled access to the information enclosed within the capsule. Encapsulation is the process of enclosing within classes and objects the attributes and the methods. But information hiding cannot be treated as encapsulation, it is different e.g. an array or a record structure also encloses the information but this information cannot be said to be hidden. It is true that the encapsulation mechanism like classes and objects hide information but these also provide visibility of some of their information through well defined interfaces.

3) Object-oriented programming languages are characterised by four essential properties: identity in terms of objects, classification in terms of classes, polymorphism and inheritance. Any other programming language that uses objects and in addition, supports at least one or more of these essential characteristics (may also support features other than these four, in addition) is called as an object-based programming language.

1.11 FURTHER READINGS

1) Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Lorensen W., *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.

2) Bolshakova E., Programming Paradigms in Computer Science Education, International Journal: *Information Theory & Applications*, 12(3), 2005.

3) http://en.wikipedia.org/wiki/Object-oriented_programming_language

4) http://en.wikipedia.org/wiki/Object-based_language

5) http://140.134.26.20/wbem/eng/ch3.html

6) http://www.desy.de/gna/html/cc/Tutorial/node2.html#SECTION00200000000000000000

7) The *C++ Programming Language* by Bjarne Stroustrup, Addison-Wesley, 3rd edition, 1997.

8) *C++ Programming Today* by Johbston Barbara Johnston 2nd Edition, PHI

9) *C++: The Complete Reference*, Herbert Schildt, 4th Edition, Mc Graw Hill.