UNIT 2 OBJECT-ORIENTED PROGRAMMING SYSTEM

Structure **Page Nos.**

2.0 INTRODUCTION

The object-oriented programming over the last decade has become a major trend in developing software and is accepted in both industry as well as research labs and academia. Object-oriented programming system (OOPS) has come a long way and has seen many languages implementing it as a way of developing software. OOPS is implemented by languages in many flavours, some are purely object-oriented (for example SMALLTALK) and some are a combination of traditional procedural object-oriented programming. OOPS have several advantages over earlier programming paradigms. In this unit, we will present a general description of the basic concepts of object-oriented programming.

Object-oriented technologies can either confuse you or make you successful. It depends on your approach of using them and your understanding of the ultimate goal of object-oriented (OO) language.

2.1 OBJECTIVES

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

- describe the concepts of object-oriented programming;
- define various terms used in object-oriented programming, and
- describe the terminology like abstraction encapsulation, inheritance, polymorphism.

2.2 WHAT IS OOPS?

Object Oriented Programming Systems (OOPS) is a way of developing software-using objects. As described in the previous unit, objects are the real world models, which are entities in themselves. That is they contain their own data and behaviour. An object resembles the physical world. When something is called as an object in our world, we associate it with a name, properties, etc. It can be called or identified by name and/or properties it bears. When these real world objects are called, they act in some or the other way. Similarly, object in OOPS are called or referenced by the way

of messages. Objects have their own internal world (data and procedures) and external interface to interact with the rest of the program (real world).
Thinking in terms of objects results from the close match between objects in the programming sense and objects in the real world. What kind of things become objects in object-oriented programs? The answer depends on your imagination, but here are some typical categories to start you thinking.

Physical Objects

ATM in Automated Teller Machines

Aircraft in an Air traffic control system

Countries in the political model

Elements of the Computer User Environment

Windows

Menus

Graphic Objects (lines, rectangles, circles)

The mouse, keyboard, disk drives, printer

Data Storage Constructs

Arrays

Stacks

Linked Lists

Binary Trees

Human Entities

Employees

Students

Customers

Let us think about an object: employee. The question that we should ask for this object design is: "What are the data items related to an Employee entity? and, What are the operations that are to be performed on this type of data?"

One possible solution for employee type may be:

Object: Employee

Data: Name, DOB, Gender, Basic Salary, HRA, Designation, Department, Contact address, qualification, any other details.

Operations: Find_Age, compute_Salary, Find_address.
Create_new_employee_object, delete_an_old_employee_object.

But now the obvious Question is: How are the objects defined?

The objects are defined via the classes.

2.3 CLASS

Objects with similar properties are put together in a class. A class is a pattern, template, or blueprint for a category of structurally identical items (objects). OOPS

programmers view Objects as instances of Class. A class is a blueprint from which objects can be created/instantiated.

Class contains basic framework, i.e., it describes internal organisation and defines external interface of an object. When we say a class defines basic framework, we mean that it contains necessary functionality for a particular problem domain. For example, suppose we are developing a program for calculator, in which we have a class called calculator, which will contain all the basic functions that exist in a real world calculator, like add, subtract multiply, etc., the calculator class will, thus, define the internal working of calculator and provides an interface through which we can use this class. For using this calculator class, we need to instantiate it, i.e., we will create an object of calculator class. Thus, calculator class will provide a blueprint for building objects. An object which is an instance of a class is an entity in itself with its own data members and data functions. Objects belonging to same set of class shares methods/functions of the class, but they have their own separate data values.

Class in OOPS contains its members and controls outside access, i.e., it provides interface for external access. The class acts as a guard and, thus, provides information hiding and encapsulation. These concepts are discussed later in the unit.

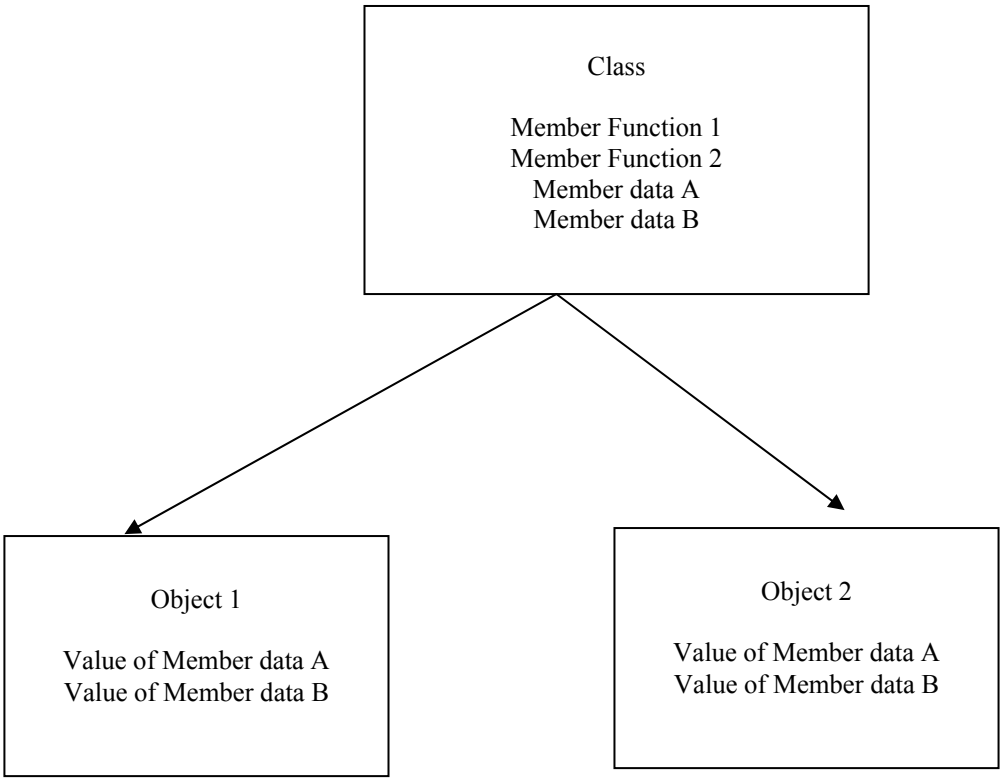

Figure 1: Class and Objects

All the objects share same member data functions, but maintain separate copy of member data. (Please refer *Figure 1*). You can use class for defining a user defined data type. A class serves a plan, or a template that specifies what data and what functions will be included in objects of that class. Defining the class does not create any objects, just as the mere existence of a type **int** does not create variables of type int.

A class is a description of a number of similar objects. A class has meaning only when it is instantiated. For example, we can use a class employee directly. We define a class employee and instantiate it.

Class Employee;

Employee John;

Now, we can have various operations on John like compute_salary of john.

☞ Check Your Progress 1

1) Which of the following cannot be put under the category of an object?

- Employers
- Manager
- Doubly linked list
- Quick sorting of numbers
- Square root of a number
- Students
- Word file

2) State True or False

a) Two objects of same class share same data values. True ☐ False ☐

b) A class contains at least one object. True ☐ False ☐

c) An instantiation of a class is an object. True ☐ False ☐

d) Objects are associated with one or more classes. True ☐ False ☐

2.4 INHERITANCE

Let us now consider a situation where two classes are generally similar in nature with just couple of differences. Would you have to re-write the entire class?

Inheritance is the OOPS feature which allows derivation of the new objects from the existing ones. It allows the creation of new class, called the derived class, from the existing classes called as base class.

The concept of inheritance allows the features of base class to be accessed by the derived classes, which in turn have their new features in addition to the old base class features. The original base class is also called the parent or super class and the derived class is also called as sub-class.

An example

Cars, mopeds, trucks have certain features in common, i.e., they all have wheels, engines, headlights, etc. They can be grouped under one base class called automobiles. Apart from these common features, they have certain distinct features which are not common like mopeds have two wheels and cars have four wheels, also cars use petrol and trucks run on diesel.

The derived class has its own features in addition to the class from which they are derived.

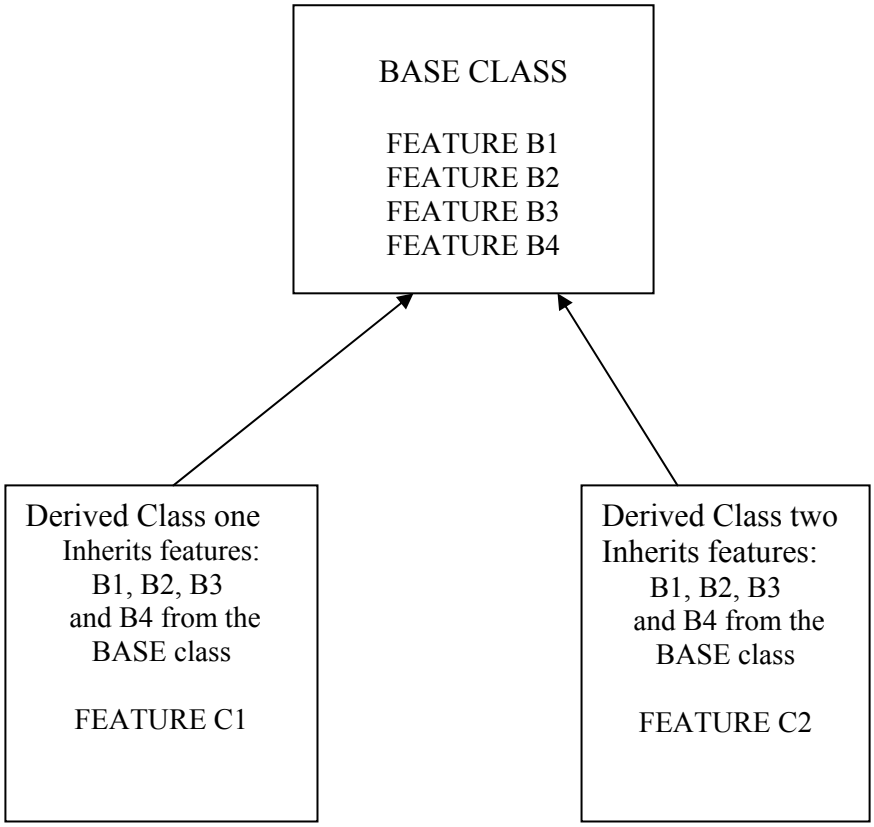

Figure 2: Inheritance

In the *Figure 2,* classes one and two are derived from base class. Note that both derived classes have their own features C1 and C2 in addition to derived features B1, B2, B3, B4 from base class.

Let us extend our example of employee class in the context of Inheritance. After creating the class, Employee, you might make a sub-class called, Manager, which defines some manager-specific operations on data of the sub-class 'manager'. The feature which can be included may be to keep track of employee being managed by the manager.

Inheritance also promotes reuse. You do not have to start from scratch when you write a new program. You can simply reuse an existing repertoire of classes that have behaviour similar to what you need in the new program.

Inheritance is of two types

Single Inheritance

When the derivation of a derived class is from one base class, it is called single inheritance.

Multiple Inheritance

When the derivation of a derived class is from more than one base classes, then it is multiple inheritance. The concept of inheritance is the same in both type of inheritance, the only difference being in number of base classes.

Advantages of Inheritance

Reuse of existing code and program functionality: The programmer does not have to write and re-write the same code for logically same problems. They can derive the

existing features from the existing classes and add the required characteristics to the new derived classes.

Much of the art of object oriented programming involves determining the best way to divide a program into an economical set of classes. In addition to speeding development time, proper class construction and reuse results in far fewer lines of code, which translates to less bugs and lower maintenance costs.

Less labour intensive: The programmers do not have to rewrite the same long similar programs just because the application to be developed has slightly different requirements.

Well organised: The objects are well organised in a way that they follow some hierarchy.

An example of inheritance could be taken from bank. A bank maintains several kind of bank accounts, e.g., Savings Account, Current Account, Loan Account, etc., all these accounts at bank have certain common features like customer name, account number, etc., at the same time, every type of account has its own characteristics, e.g., loan account could have guarantors name, and an operation for periodic repayment of loan; savings account could contain introducers name and an operate-account operation etc. All these accounts can be derived from the base class Bank Account and have separate derived classes for each of them.

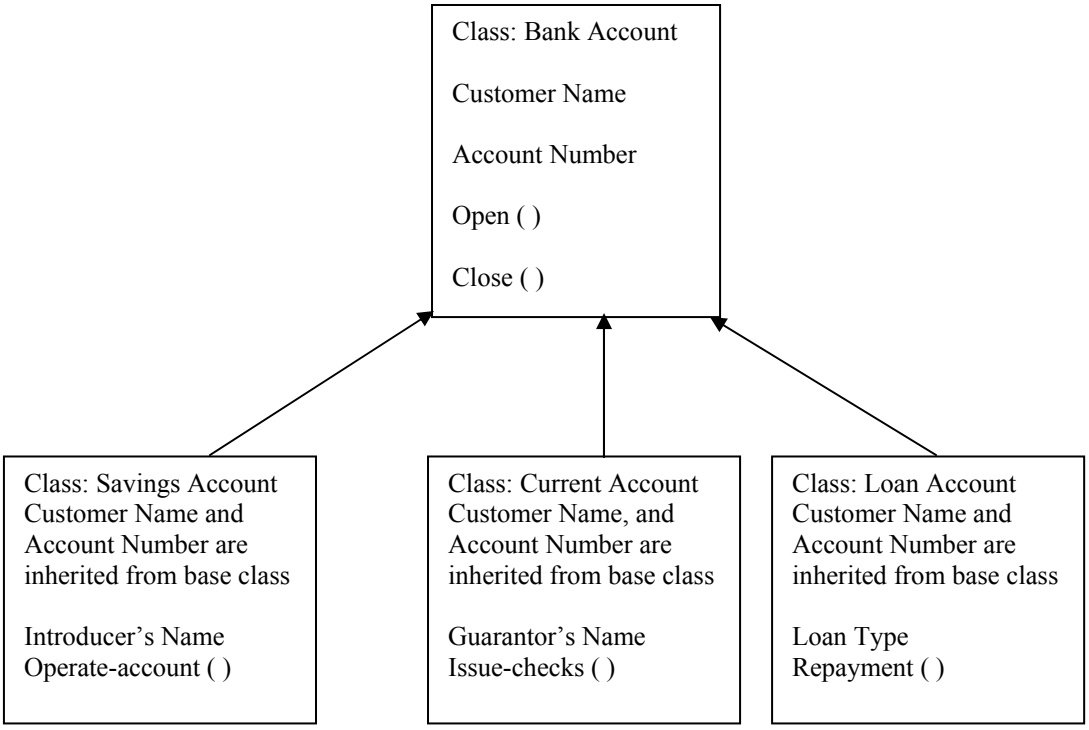

Figure 3: Example of Inheritance

2.5 ABSTRACTION

Whenever we have to solve a problem then first we try to distinguish between the important and unimportant aspects of the problem. This is abstraction, thus, Abstraction identifies pattern and frameworks, and separate important and non-important problem spaces. There could be many levels of abstractions, like,

- Most important details,
- Less important details, and

- Unimportant details.

To invent programs, you need to be able to capture the same kinds of abstractions as the problem have, and express them in the program design.

From the implementation point of view, a programmer should be concerned with "what the program is composed of and how does it works?" On the other hand, a user of the program is only concerned with "What it is and what it does."

A programming language should facilitate the process of a program invention and design by letting you encode abstractions to reveal the way things work. If the language inherently does not support Abstraction, then it is task of the programmer to implement Abstraction. All programming languages provides a way to express Abstractions. In essence, Abstraction is a way of grouping implementation details, hiding them, and giving them, at least to some extent, a common interface.

Abstraction in a procedural language

The principal units of Abstractions in the C language are structures and functions. Both, in different ways, hide elements of the implementation. For example, C structures group data elements into larger units, which can then be handled as a single entity. One structure can include others, so a complex arrangement of constructs can be built from simpler structure constructs. A structure is an example of data Abstraction.

Functions in C language encapsulate behaviours that can be used repeatedly. Data elements local to a function are protected within their own domain. Functions can reference (call) other functions, so quite complex behaviours can be built from smaller pieces. Functions in C language represent procedural side of Abstraction.

Well-designed functions are reusable. Once defined, they can be called any number of times. The most useful functions can be collected in libraries and reused in different applications. All the user needs is the function interface and not the source code. Each function in C must have a unique name. Although the function may be reusable, its name is not.

C structures and functions are able to express Abstractions to certain extent, however, they maintain the distinction between data and operations on data.

Abstraction in an Object-Oriented language

Suppose, you have a group of functions that can act on a specific data structures. To make those functions easier to use by, you can take the data structure out of the interface of the entity/object by supplying a few additional functions to manage the data. Thus, all the work of manipulating the data structure, viz., allocating data, initialising, output of information, modifying values, keeping it up to date, etc., can be done through the functions. All that the users do is to call the functions and pass the structure to them.

With these changes, the structure has become an opaque token that other programmers never need to look inside. They can concentrate on what the functions do, not how the data is organised. You have taken the first step toward creating an object.

The next step is to provide support to Abstraction in the programming language and completely hide the data structure. In such implementations, the data becomes an internal implementation details; and user only sees the functional interface. Because an object completely encapsulates their data (hide it), users can think of them solely in terms of their behaviour.

The hidden data structure unites all of the functions that share it. So, an object is more than a collection of random functions; it is a grouping, a bundle of related behaviours that are supported by shared data.

This progression from thinking about functions and data structures to thinking about object behaviour is the essence of object-oriented programming. It may seem unfamiliar at first, but as you gain experience with object-oriented programming, you will find that it is a more natural way to think about things. By providing higher level of Abstractions, object-oriented programming language give us a larger vocabulary and a richer model to program in.

Mechanisms of Abstraction

Thus, Abstraction is when we create an object we concentrate only on its external working while discarding unnecessary details. The internal details are hidden inside the object, which makes an object abstract. This technique of hiding details is referred to as data abstraction.

Objects in an object-oriented language have been introduced as units that implement higher-level abstractions and work as coherent role-players within an application. However, they could not be used this way without the support of various language mechanisms. Two of the most important mechanisms are: Encapsulation, and Polymorphism.

☞ Check Your Progress 2

1) What is the need of inheritance?

……

……

……

……

2) What is Abstraction in the context of object oriented programming?

……

……

……

……

3) State True (T) or False (F)

a) Abstraction can be implemented using structures.
True ☐ False ☐

b) Multiple Inheritance means that one base class have multiple sub-classes. True ☐ False ☐

c) Inheritance is useful only when it is single inheritance
True ☐ False ☐

d) Object-oriented languages provide higher level of abstraction, that is, they are closer to real world objects. True ☐ False ☐

2.6 ENCAPSULATION AND INFORMATION HIDING

To design effectively at any level of abstraction, you should not be involved too much in thinking about details of implementation, rather you should be thinking in terms of units for grouping those details under a common interface.

For a programming unit to be truly effective, the barrier between interface and implementation must be absolute. The interface must encapsulate the implementation; hide it from other parts of the program. Encapsulation protects an implementation from unintended actions and inadvertent access.

In programming, the process of combining elements to create a new entity is encapsulation. For example, a procedure is a type of encapsulation because it combines a series of computer instructions. Its implementation is inaccessible to other parts of the program and protected from whatever actions might be taken outside the body of the procedure. Likewise, a complex data type, such as a record or class, relies on encapsulation. Object-oriented programming languages rely heavily on encapsulation to create high-level objects.

In an Object-oriented language, a class is clearly encapsulated as the data variables and the related operations on data are placed together in a class.

For example, in a windows based software, the window object contains Window's dimensions, position, colour, etc. Encapsulated with these data are the functions which can be performed on Window, i.e., moving, resizing of Window, etc. The other part of this window program will call upon window object to carry out the necessary function. The calling or interacting with the action will be performed by sending messages to it. The required action will be performed by the window object according to its internal structure. This internal working is hidden from the external world or from the other part of the software program.

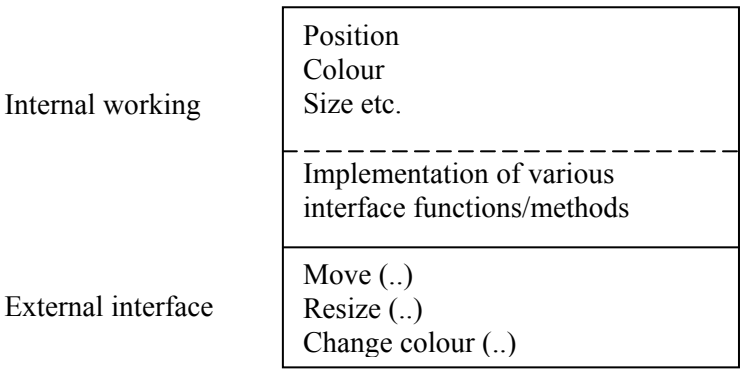

Figure 4: Window Object

Based on the requirement and choice in general, there are three basic types of access modes of the members of class:

The data variables of the class A can only be accessed by only the functions of the class A – this is private mode of access.

The data variables of the class A can be accessed by any function– this is public mode of access.

The data variables of the class A can only be accessed by the functions with some special privileges – this is protected mode of access.

Thus, an object's variables are hidden inside the object and invisible outside it. The encapsulation of these instance variables is sometimes also called information hiding. The process of hiding details of an object or function is information hiding. Information hiding is a powerful programming technique because it reduces complexity. One of the chief mechanisms for hiding information is encapsulation – combining elements to create a larger entity. The programmer can then focus on the new object without worrying about the hidden details. In a sense, the entire hierarchy of programming languages – from machine languages to high-level languages – can be seen as a form of information hiding.

Information hiding is also used to prevent programmers from changing – intentionally or unintentionally – parts of a program.

It might seem, at first, that hiding the information in instance variables would constrain your freedom as a programmer. Actually, it gives you more room to act and frees you from constraints that might otherwise be imposed. If any part of an object's implementation could leak out and become accessible or a concern to other parts of the program, it would tie the hands of both, the persons who have implemented the Object and of those who would use the object. Neither could make modifications without first checking with the other.

For example, you are interested in developing the object say "Pump" for the program that models use of water and you want to incorporate it in another program you are writing. Once the interface to the object is decided, you do not have to be concerned about fixing the bugs, and finding better ways to implement it, without worrying too much about the people using it.

You will decide all the functions and operations of pump and you will be providing a complete functional object, i.e., pump for the other program through an interface. The programmer of another program will solely depend on the interface and will not be able to break your code and change the functionality and implementation of the object pump. As a matter of fact, rather s/he will not be even knowing about the implementation details of the object pump. S/he will simply be knowing the functional details of the object pump. Your program is insulated from the object's implementation. This way the information about the object pump will be hidden from all other modules except the one of which it is part.

Moreover, although those implementing or using the object pump would be interested in how you are using the class and might try to make sure that it meets your needs, they do not have to be concerned with the way you are writing your code. Nothing you do, can touch the implementation of the object or limit their freedom to make changes in future releases. The implementation is insulated from anything that you or other users of the object might do.

2.7 POLYMORPHISM

The word polymorphism is derived from two Latin words poly (many) and morphs (forms). This concept of OOPS provides one function to be used in many different forms depending on the situation it is used. The polymorphism is used when we have one function to be carried out in several ways or on several object types. The polymorphism is the ability of different objects to respond in there own ways to an identical message.

When a message is sent requesting an object to do a particular function, the message names the function the object should perform. Because different objects can have different functions with the same name, the meaning of a message must be decided with respect to the particular object that receives the message. Thus, the same message sent to two different objects can invoke two different functions.

The main advantage of polymorphism is that it simplifies the programming interface. It allows creating of conventions that can be reused from class to class. Instead of inventing a new name for each new function, you add to a program, the same names may be reused. The programming interface can be described as a set of abstract behaviours that may be different from the classes that implement them.

Overloading

The terms "polymorphism" and "argument overloading" refer basically to the same thing, but from slightly different points of view. Polymorphism takes a pluralistic point of view and notes that several classes can have a method with the same name. Argument overloading takes the point of the view of the function name and notes that it can have different effects depending on what kind of object it applies to.

Operator overloading is similar. It refers to the ability to turn operators of the language (such as '==' and '+' in C) into methods that can be assigned particular meanings for particular kind of objects.

For example, we need to build a program, which will be used for addition. The input to the program should not depend on input variable, that is, it should be able to produce the result of addition whether the input is of integer or float or character variable.

The polymorphism allows the objects to act as black boxes, i.e., they have common external interface which will allow them to be called or manipulated in the same way. But their internal working and the output is different from each other which depends on the way in which they are invoked or manipulated.

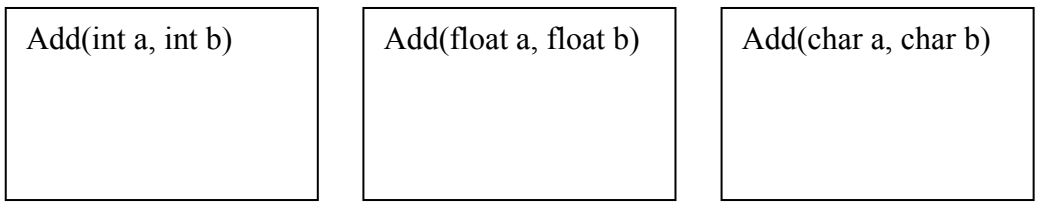

Figure 5: Example of polymorphism

These addition functions act as black boxes. All of them do the same thing and bear the same name, but perform differently, depending on the arguments passed to them.

Polymorphism also permits code to be isolated in the function of different objects rather than be gathered in a single function that enumerates all the possible cases. This makes the code you write more extensible and reusable. When a new case comes along, you do not re-implement existing code, but only add a new class with a new function, leaving the code that is already written alone.

A very common example for the above is using the "draw" function of an object. We might have to draw a circle, or square or triangle, etc. But for different drawing methods, we will not be creating different methods like draw-Circle or draw-Square, etc., rather we will be defining draw methods with appropriate arguments.

For example,

- draw (float radius) – Call to this kind of argument will draw circle….e.g.,
- draw (float 5.2) will draw circle with radius 5.2.
- draw (float a, float b), will draw square or rectangle.

The same can be extended to sub-classes also where from a base class shape, sub-classes such as circle, square and rectangle can be created. Each sub-class will have its own implementation for the base class function draw (). Thus, enabling drawing of circle or square or rectangle based on the sub-class that envokes the message. Thus, polymorphism is a very strong mehanism that supports reuse of similarities among object hiding dissimilarities under different behaviour as a result of the same message to different objects. The implementation level details about polymorphism is given in Block 2.

☞ Check Your Progress 3

1) What is operator overloading? Is it different from polymorphism?

……

……

……

2) State True or False

a) Encapsulation involves data hiding. True ☐ False ☐

b) A window-based environment has variables like position, colour, size that can be modified by any function. This is a valid example of information hiding. True ☐ False ☐

c) Information hiding increases maintenance-related problems. True ☐ False ☐

d) Polymorphism can be implemented through any object-oriented programming language. True ☐ False ☐

2.8 SUMMARY

In this unit, an overview of various concepts relating to object-oriented system has been presented. The base of all the concepts in object oriented programming is the "object". All the concepts are related to either its properties or behaviour. Class defines the behaviour and the data members of an object. An object encapsulates its data (generally) and generally external interface is the only media for communication with the objects. Inheritance and polymorphism are the concepts that have given major advantages to object oriented programming. Thus, the basic concepts discussed in this unit can be considered as the basic strengths of object-oriented programming system.

2.9 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Quicksorting of numbers; square root of numbers.

2) a) False
b) False
c) True
d) False

Check Your Progress 2

1) Inheritance helps in representing
   1) Classes in a hierarchy: Well-organised problem solution space.
   2) Extending existing classes thus, promotes reuse.
   3) Reducing the duplication of efforts.

2) Abstraction is to create model of behaviour of a real object and hiding its internal working details.

3) a) False
   b) False
   c) False
   d) True

Check Your Progress 3

1) Operator overloading is the way of defining the meaning to operator for a class by using methods/functions for operators. For example, the function for operator + can be written that defines concatenation of two string objects. Polymorphism is a generic concept that involves operator and function overloading. It can be used across several classes where the same function name handle may be used for different types of object.

2) a) True
   b) False
   c) False
   d) True