UNIT 3 ADVANCED CONCEPTS

Structure Nos. **Page**

3.0	Introduction	29
3.1	Objectives	29
3.2	Dynamism	29
	3.2.1 Dynamic Typing	
	3.2.2 Dynamic Binding	
	3.2.3 Late Binding	
	3.2.4 Dynamic Loading	
3.3	Structuring Programs	33
3.4	Reusability	34
3.5	Advantages of Object-Oriented for Project Design and Development	35
3.6	Summary	37
3.7	Solutions/Answers	37

3.0 INTRODUCTION

Object-oriented design has become popular in the industry. In the previous two Units of this Block, our focus was on discussion of some of the basic concepts of object-oriented programming. However, in this Unit, we would like to touch upon some of the advanced concepts that are useful for implementation of object-oriented programming paradign. This Unit includes discussions on dynamism, the basic for overloading and polymorphism, the reusability that is the plank for building reusable classes using standards/tool like CORBA, COM/DCOM, etc., and software project related issues. However, in this Unit, our objective is not to touch issues relating to object-oriented modeling and design that will be covered in the last Unit of the Block.

3.1 OBJECTIVES

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

- define the dynamism and its implications in the context of overloading polymorphism;
- describe the reusability concepts, that is bad to better class design, and
- define the features of object-oriented projects.

3.2 DYNAMISM

Historically, the amount of memory allocation was determined at the compile and link time of the program. The source code size was fixed and generally the memory allocated to a program was as per the source code and could not be increased or decreased once memory is allocated. The approach was restrictive in nature. It not only put the restriction of maximum size of the code for various programming languages, but also the program design and programming techniques being followed. For example, a function for allocation and de-allocation of memory could not be thought of for such a system. However, development in hardware and software technology and with functions (like malloc(), new) that dynamically allocate memory as a program runs opened possibilities that did not exist before.

Compile-time and link-time constraints are limiting in nature as they force resource allocation to be decided from the source code, rather than from the information that is obtained from the running program.
Although dynamic allocation removes one such constraint that is allocating memory at run time to a data object, many others constraints equally as limiting as static

memory allocation, remain. For example, the objects that make up an application must be bound to data types at compile time and the boundaries or size of an application are typically fixed at link time. The above constraints necessitates that every part of an application must be created and assembled in a single executable file. New modules and new types cannot be introduced as the program is being executed.

Object-oriented programming system tries to overcome these constraints and try to make programs as dynamic as possible. The basic idea of dynamism is to move the burden of decision making regarding resources allocation and code properties from compile time and link time to run time. The underlying philosophy is to allow the control of resources indirectly rather than putting constraints on their actions by the demands of the computer language and the requirements of the compiler and linker. In other words, freeing the world of users from the programming environments. But, what is dynamism in the context of object-oriented design?

There are three kinds of dynamism for object-oriented design. These are:

- Dynamic typing: That is, to determine the class of an object at run time.
- Dynamic binding: That is, the decision object involving a function in response to a call is moved to run time.
- Dynamic loading: That is, adding new components to a program as it is getting executed.

Let us discuss them in more detail in the following sub-sections.

3.2.1 Dynamic Typing

In general, the compilers give an error message if the code assigns a value to a variable of a different type that it cannot accommodate. Some typical warning messages in such cases may be as under:

> "incompatible types in an assignment".
>
> "assignment of integer from pointer does not have a cast".

Type checking at compile time is useful as it tries to catch many expressions related errors, however, there are times when it can interfere with the benefits of mechanisms like polymorphism or the mechanisms where the type of an object is not known till run time.

Suppose, you want to send an object a message to perform the print method. As the case with other data elements, the object is also represented by a variable. In case the class of this variable is to be determined at compile time, then it will not be possible to change the decision about what kind of object should be assigned to the variable at run time. Once class of variable is fixed in source code, then it automatically fixes the version of print method that is to be invoked in response to the message.

However, if we delay the assignment discovery of a class/type of a variable, then it will provide a very flexible approach of class-to-type binding and results in assignment of any kind of object to that variable. Thus, depending on the class of the receiver object, the print message might invoke different versions of the method and produce very different results at run time.

Dynamic typing is the basis of dynamic binding. (Please refer to next sub-section). However, dynamic typing does more than that. It allows associations between objects to be determined at run time, rather than fixing them to be encoded in static compile time. For example, a message could pass an object as an argument without declaring exactly the type/class of that object. The message receiver may then send a message to the object again without ascertaining the class of the object. Because the receiver uses the object to do some of its own work, which is in a sense customised

by the object of indeterminate type (indeterminate in source code, but, not at run time).

3.2.2 Dynamic Binding

In standard C program, one can declare a set of alternative functions For example, the standard string-comparison functions,

```
int strcmp (const char *,  const char *); /* case sensitive */
int strcscmp (const char *, const char *); /* case insensitive*/
```

Let us declare a pointer to function sting_compare that has the same return and argument types:

```
int (* string_compare) (const char *, const char *);
```

In the above case, you can determine the function assignment to pointer at run-time using the following code (through command line arguments):

```
if (**argv == 'i')

string_compare = strcscmp;

else

string_compare = strcmp;
```

Thus, you can call the function through the pointer:

```
if (string_compare (s1, s2))
```

This is a static time binding, where the procedures will be bound at compile time but for a string, which function to follow will be determined at program run time.

This is quite close to what in object-oriented programming is called dynamic binding. Dynamic binding means that delaying the decision of exactly which method to perform until the program is running.

Dynamic binding is not supported by all object oriented languages. It can easily be accomplished through messaging. You do not need to go through the indirection of declaring a pointer and assigning values to it as shown in the example above. You also need not assign each alternative procedure a different name.

Messages are used for invoking methods indirectly. Every message must match a method implementation. To find the matching method, the messaging system must check the class of the receiver and locate the implementation of the method requested in the message. When such binding is done at run time, the method is dynamically bound to the message. When it is done by the compiler, then the method is statically bound.

Dynamic binding is possible even if of dynamic typing does not exist in a programming language but it is not very useful. For example, the benefits of waiting until run time to match a method to a message when the class of the receiver is already fixed and known to the compiler are very little. The compiler could just as well find the method itself; such results will not be different from run-time binding results.

However, in case of the type/class of the receiver is dynamic, the compiler cannot, determine which method to invoke. The method can be bound only after the class of the receiver is bounded at run time. Thus, Dynamic typing, entails dynamic binding.

Dynamic typing opens the possibility that a message might have very different results depending on the class of the receiver because the run-time data may influence the outcome of a message.

Dynamic typing and binding opened the possibility of sending messages to objects that have not yet been designed. If object types need not be decided until run time, you can give more freedom to designers to design their own classes and name their own data types, and still your code may send messages to their objects. All you need to decide jointly is the message, that is, the interfaces of the objects and not the data types.

3.2.3 Late Binding

Some object-oriented programming languages (such as C++) require a message receiver to be statically typed in source code, but do not require the type to be exact as per the following rule:

"An object can be typed to its own class or to any class that it inherits from." The compiler, therefore, cannot differentiate whether a message receiver is an instance of the class specified in the type declaration, OR, an instance of a sub-class, OR, an instance of any further derived class. Since the sender of the message does not know the class of the receiver, it does not know which version of the method named in the message will be invoked.

As a message is received by a receiver, it can either receive it as an instance of the specified class and the class can simply bind the method defined for that class to the message. As an alternative, the binding may be delayed to run-time. In C++ the binding decisions are delayed to run time for the methods, also called member functions, which are in the same inheritance hierarchy.

This is referred to as "late binding" rather than "dynamic binding". It is "dynamic" in the sense that it happens at run time, however, it carries with it strict compile-time type constraints, whereas "dynamic binding" as discussed earlier is unconstrained.

3.2.4 Dynamic Loading

From the von Neumann architecture days, the general rule for running a program is to link all its parts together in one file the entire program is loaded into memory that may be virtual memory, prior to execution.

Some object-oriented programming environments have overcome this constraint. They allow different parts of an executable program to be kept in separate files. The program can be created from the bits and pieces as and when they are needed. The component of the programs are dynamically loaded and linked with the rest of program at run-time. The various facilities executed by a user determine the parts of the program that are to be kept in memory for execution purpose.

This is a very useful concept as, in general, only a small core of a large program is used by the users. Thus, only standard core program may be loaded in the memory for execution. Other modules may be called as per the request of the user. Thus, the component/sections of program that are not executed at all are also not loaded in the memory.

Dynamic loading raises some interesting possibilities. For example, it encourages modular development with an added flexibility that the entire program may not to be developed before a program can be used. The programs can be delivered in pieces and you can update one part of it at any time. You can also create program that groups many different tools under a single interface, and load just the tools desired by the user. In addition, alternative sets of tools may also be offered for the given task,

the user can select any one tool from the available sets. This will cause only the desired tool set to be loaded.

As per present one of the important benefits of dynamic loading is that it makes applications extensible. A program designed by you can be added or customized as per your needs. Such examples are common when we look into operating systems like Linux and it utilities. Such program must provide some basic framework for extension at run time these program find the pieces that have been implemented and load them dynamically.

One of the key requirements of dynamic loading is to make a newly loaded part of a program to work with parts already running, especially when the different parts of the program are written by different people. However, such a problem does not exist in an object-oriented environment because code is organised into logical modules with a clear distinction between implementation and interface. When classes are dynamically loaded, the newly loaded code can not clash with the already loaded code. Each class encapsulates its implementation and has an independent name space.

In addition, dynamic typing and dynamic binding concepts allows classes designed by others to fit effortlessly into your program design. Your code can send messages to other's objects and vice versa neither of you has to know what classes the others have implemented. You only need to agree on the communication protocol.

Loading and Linking

Dynamic loading also involves dynamic Linking of programs that requires various parts to be joined so that they can work together. The program is loaded into volatile memory at run time. Linking usually precedes loading. Dynamic loading involves the process of separately loading new or additional parts of the program and linking them dynamically to the parts of the program already running.

3.3 STRUCTURING PROGRAMS

An object-oriented program has two basic kinds of structure:

The first structure is in the inheritance hierarchy of class definitions.
The other structure is the pattern of message passing as the program runs. These messages create a network of object connections.

The inheritance hierarchy determines how objects are related by type. For example, in the program that models workers in an organisation, it might turn out that managers and employees are of the same kind of object, except that manager control a group of employees. Their similarities can be captured in the program design if the manager and employee classes inherit from a common class: Human Resources.

The network of object connections explains the working of the program. For example, manager objects might send message requesting employee to do a piece of work. Employee object might communicate with the tools objects, etc. To communicate with each other in this way, objects must know about each other's existence. These connections define a program structure.

Thus, the object-oriented programs are designed by laying down the network of objects with their behaviours and basic patterns of interaction, and finally by arranging the hierarchy of classes. The object-oriented programming requires structuring, both in the activities of the program and its definition in terms of inheritance hierarchy of classes.

3.4 REUSABILITY

One of the major goals of object-oriented programming is to make reusable programs to the extent possible such that it can be used in many different situations and applications without re-implementation, even though it may be in slightly different form, from the earlier use.

Reusability of a program is influenced by a number of factors, such as:

- Reliability of the code, whether it is bug free or not
- Clarity of documentation
- Simplicity of programming interface
- Efficiency of code
- The richness of feature set of an object to cater for many different situations.

These factors can also be used to judge the reusability of any program irrespective of the language of implementation as well as class definitions. For example, efficient and well documented programs/functions would be better from the reusability point of view in comparison to the programs that are undocumented and unreliable.

This class definitions lend themselves to reusable code in a much better form than that of procedural functions. Functions can be made more reusable by passing data as arguments rather than using global variables. However, the reusability of functions is still constrained as per the following reasons:

- Function names are global variables by themselves. Each function must have a unique name. This makes it difficult to rely heavily on library code when building a complex system. It makes the programming very extensive and, thus, hard to learn and difficult to generalise. On the other side, classes can share programming interfaces. With the same naming conventions used over and over again, a great deal of functionality can be packaged with a relatively small and easy-to-understand interface because of various concepts like inheritance, overloading and polymorphism.

- The second problem with functions is that they are selected from a library one at a time. The programmer needs to pick and choose the individual functions as per his needs. In contrast, objects are the packages of functionality and not just individual methods and instance values. They provide integrated services. Thus, an object-oriented library does not have functions that are to be joined by user for a solution. They have objects, which represent a solution to a problem.

- Functions are typically coupled to particular kinds of data structures for a specific program. The interaction between the data and function is major part of the interface. A library function is useful only to those who are using the same kind of data structures. However, an object hides the data, therefore, does not face such a problem. This is one of the main reasons why classes can be reused more easily than functions.

The data of an object is protected by access rights and cannot be altered by any other part of the program. Methods of a class are therefore responsible for integrity of data of an object. The external access cannot put an illogical or untenable state to data of an object. This makes an object data structure more reliable than that of the data passed to a function. Therefore, methods can rely on such reliable data and hence the reusable methods are easier to write.

Moreover, because the data of an object is hidden from an external user, a class can be redesigned to use a better data structure without affecting its interface. All programs that use the changed class can use the new version without changing any source code; no re-programming is required.

3.5 ADVANTAGES OF OBJECT-ORIENTED FOR PROJECT DESIGN AND DEVELOPMENT

Object-oriented programming allows restructuring of the program design which allows thinking about a software at the top level and not benefit at low-level implementation details. The main advantages of object-oriented programming project development are code reusability, complexity controls by a co-operating group, etc. Let us look into these advantages in more details.

Design at high-level abstraction

A program designed at high level of abstraction allows easier division of labour on logical lines; a project organisation grows out of this design.

An object design allows to focus on common goals, instead of losing them during implementation. Object design also helps in visualizing working of module in the complete program. Their collaborative efforts are, therefore, likely to be organised and oriented towards problem solution.

Separate interface and implementation moves the details to design stage.

The inter connections among various components of an object-oriented program are normally worked out early in the design process. The interactions must be well defined prior to implementation.

During implementation phase only the interface of object needs to be monitored for. Since each class encapsulates its implementation and has its own name space, the object-oriented projects need not be coordinated for implementation.

Modularising the project in manageable components

The object-oriented system support modularity, which implies that a software can be broken into its logical components. Each of these logical components can be implemented separately. Thus, software engineers may be asked to work on different class or module separately.

The benefit of modularisation is not only in implementation, but also at the maintenance time. The class boundaries contain the problems that are related to a class. Thus, any bug can be tracked to a class and can be rectified.

An interesting outcome with respect to separating responsibilities by class is that each part can be worked on by specialist object. Classes can be updated periodically for performance optimisation and as per the new technologies. Such updates need not be coordinated with other parts of the object-oriented program. The improvements to a class implementation can be made at any time if its interface remains unchanged.

Simple interfaces to remember (for example, windows)

The mechanisms such as polymorphism in object-oriented programming yields simpler programming interfaces, since it allows the same names and conventions to be reused in any number of different classes. Thus, object-oriented classes are easy to learn, and provide a greater understanding of the working of complete system, and an easier cooperation and collaboration mechanism for program development. One such object-oriented interface commonly used by all of us is windows.

Dynamic decisions makes the program robust

Most of the object-oriented programming languages allow binding decisions dynamically that is at run time, therefore, less information needs to be available at

compile time (in source code) for allowing coordination between two objects/classes. Thus, there is less to go wrong.

Code reuse through inheritance of generic code

Inheritance in a way allows code reuse. A good practice in object-oriented system may be to create classes as specialization of more generic classes. This simplifies programming. It also simplifies the design as inheritance hierarchy describes the relationships among the different levels of classes.

Inheritance also helps in the reliability of code. For example, the code in a super class is tested by all its sub-class, thus, is tested thoroughly and hence may be most reliable.

Reuse of Tested Code

You must reuse the code as much as possible. This reduces the time of a software development cycle and you need not start a project from scratch. Object oriented classes are designed keeping reusability in mind. This also enhances the collaboration among the programmers working in different places of different organisations.

Classes available in an object-oriented library may make substantial contribution to your software. This allows you to concentrate on what you do the best and leave other tasks to the library creators. This also results in faster project completion with less effort.

An interesting facet in OOPS is that the increased reusability of object-oriented codes also increases its reliability. This is due to reuse of class in different situation; because the reused classes get tested in different situations and applications. The bugs of such classes must have already been diagnosed and fixed.

☞ Check Your Progress

1) What is dynamic typing? If a language does not support dynamic typing, then is it advisable to have dynamic binding?

   ……..

   ……..

   .

   ……..

   .

2) State True or False

   a) Late binding is the same as dynamic binding
      True ☐ False ☐

   b) Late binding is the basis of run-time polymorphism in C++
      True ☐ False ☐

   c) Dynamic loading does not involve dynamic linking
      True ☐ False ☐

   d) Objects are reusable as they support polymorphism.
      True ☐ False ☐

   e) Object-oriented programming supports better reliability also.
      True ☐ False ☐

3.6 SUMMARY

In this Unit, our attempt was to cover some of the important object-oriented features in some more details. The features which have been covered include dynamic typing, dynamic binding and late binding. These concepts are very important from the viewpoint of providing flexibility in program design and development. The other concepts which have formed the basis of popularity of object-oriented programming paradigm are its reusability, reliability and modifiability. The unit discusses why object-oriented programs are more reusable, reliable and modifiable/maintainable.

3.7 SOLUTIONS/ANSWERS

Check Your Progress

1) Dynamic typing is the determination of type of an object at run-time. It is not useful to have dynamic binding without dynamic typing as if types are bound then function indirectly get bound to a type/class to which they will be bound at run-time (dynamic binding).

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