UNIT 1 OBJECT ORIENTED METHODOLOGY-1

Structure **Page Nos.**

1.0 INTRODUCTION

Since the invention of the computer, many approaches of program development have evolved. These include modular programming, top-down programming, bottom-up programming and structured programming. The primary motivation in each case has been the concern to handle the increasing complexity of programs to be reliable and maintainable. These techniques became popular among programmers in 1970s and 1980s.

Due to the popularity of C language, structured programming became very popular and was the main technique of the 1980s. Later this technique also failed to show the desired performance in terms of *maintainability, reusability and reliability*.

As a result of this realisation, a new methodology known as Object oriented programming emerges. This approach to program organization and development attempts to eliminate some of the pitfalls of conventional programming by incorporating the best of the structured programming features with several powerful new concepts. This approach speeds the development of new programs, and, if properly used, improves the maintenance, reusability, and modifiability of software.

So, the major concern for all of us is to know what it is. What are the main features of this approach? How is it better than other approaches? What are the languages which support its various features?

In this unit, we will start with a brief discussion of the manner in which different languages have been developed so as to understand where an Object Oriented programming language fits in. Subsequently, we compare the Object Oriented approach with the procedure-oriented approach. We will also introduce the basic concepts and terminology associated with the Object Oriented (OO) approach. Finally we will talk about common OO languages and applications of OOP in various problem domains.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- find the importance of OO approach;
- define the basic concepts of OO approach;
- differentiate between object and procedure-oriented approaches;
- know about various OO languages;

- describe the applications of OOP, and
- understand the benefits of OO approach.

1.2 PARADIGMS OF PROGRAMMING LANGUAGES

The term *paradigm* describes a set of techniques, methods, theories and standards that together represent a way of thinking for problem solving. According to [Wegner, 1988], paradigms are "**patterns of thought for problem solving".**

Language paradigms were associated with classes of languages. First the paradigms are defined. Thereafter, programming languages according to the different paradigms are classified. The language paradigms are divided into two parts, **imperative** and **declarative** paradigms as shown in the *Figure 1*. Imperative languages can be further classified into **procedural** and **object oriented** approach. Declarative languages can be classified into **functional languages** and **logical languages.** In *Figure1* the examples of languages in each category are also given.

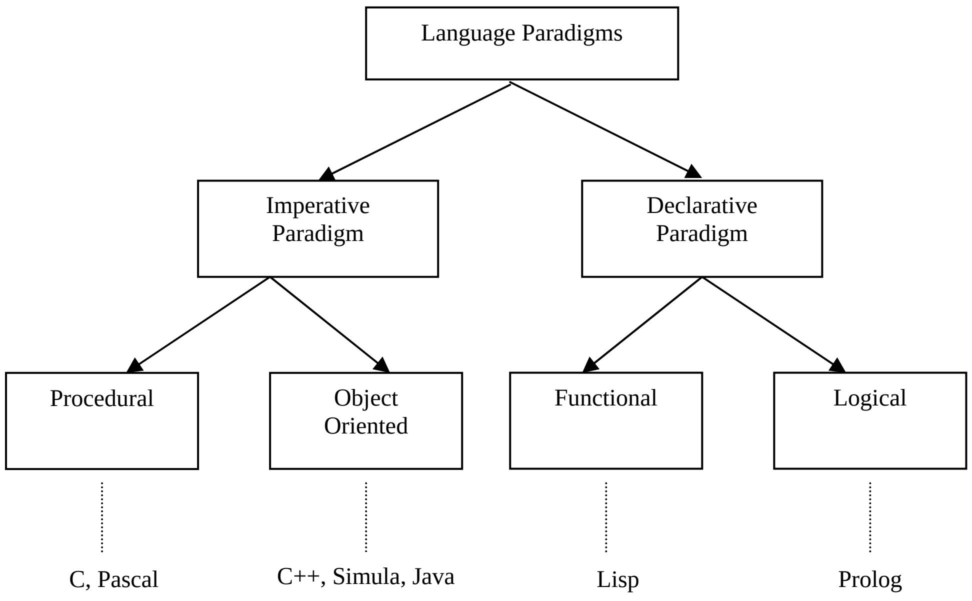


Figure 1: Language Paradigms

Imperative paradigm: The meaning of imperative is **"expressing a command or order"**, so the programming languages in this category specify the step-by-step explanation of command. Imperative programming languages describe the details of **how** the results are to be obtained, in terms of the *underlying machine model*. The programs specify step by step the entire set of transitions that the program goes through. The program starts from an initial state, goes through the transitions and reaches a final state. Within this paradigm we have the procedural approach and Object Oriented approach.

Procedural paradigm: Procedural languages are **statement oriented** with the variables holding values. In this language the execution of a program is modeled as a series of *states of variable* locations. We have two kinds of statements. **Non-executable** statements allocate memory, bind symbolic names to absolute memory locations, and initialize memory. **Executable** statements *like* computation, control flow, and input/output statements. The popular programming languages in this category are *Ada, Fortran, Basic, Algol, Pascal, Cobol, Modula, C, etc.*

Object Oriented paradigm: The Object Oriented paradigm is centered on the concept of the object. Everything is focused on objects. Can you think *what is an* *object?* We will discuss the concept of object in detail in further sections. In this language, program consists of two things: first, a set of objects and second the way they interact with each other. Computation in this paradigm is viewed as the simulation of real world entities. The popular programming languages in this paradigm are C++, Simula, Smalltalk and Java.

Declarative paradigm: In this paradigm programs declare or specify what is to be computed without specifying how it is to be achieved. Declarative programming is also known as *Value-oriented programming*. Declarative languages describe the relationships between variables in terms of functions and inference rules. The language executor applies a fixed method to these relations to produce a desired result. It is mainly it is used in solving artificial intelligence and constraint-satisfaction problems. Declarative paradigm is further divided into two categories, **functional** and **logical** paradigms.

Functional paradigm: In this paradigm, a program consists of a **collection of functions**. A function just computes and returns a value. A program consists of calling a function with appropriate arguments, but any function can make use of other functions also. The main programming languages in this category are **Lisp, ML, Scheme**, and **Haskell.**

Logic paradigm: In this paradigm programs only explain what is to be computed not how to compute it. Here program is represented by a set of relationships, between objects or property of objects known as predicate which are held to be true, and a set of logic/clauses **(**i.e. if A is true, then B is true**)**. Basically logic paradigm integrates data and control structures. The **Prolog** language is perhaps the most common example. **Mercury** language is a more modern attempt at creating a logic programming language.

1.3 EVOLUTION OF OO METHODOLOGY

The earliest computers were programmed in **machine language** using **0** and **1**. The mechanical switches were used to load programs. Then, to provide convenience to the programmer, **assembly language** was introduced where programmers use **pneumonic** for various instructions to write programs. But it was a tedious job to remember so many pneumonic codes for various instructions. Other major problem with the assembly languages is that they are machine architecture dependent.

To overcome the difficulties of Assembly language, **high-level languages** came into existence. Programmers could write a series of English-like instructions that a compiler or interpreter could translate into the binary language of computers directly.

These languages are simple in design and easy to use because programs at that time were relatively simple tasks like any arithmetic calculations. As a result, programs were pretty short, limited to about a few hundred line of source code. As the capacity and capability of computers increased, so did the scope to develop more complex computer programs. However, these languages suffered the limitations of reusability, flow control (only goto statements), difficulty due to global variables, understanding and maintainability of long programs.

Structured Programming

When the program becomes larger, a single list of instructions becomes unwieldy. It is difficult for a programmer to comprehend a large program unless it is broken down into smaller units. For this reason languages used the concept of functions (or subroutines, procedures, subprogram) to make programs more comprehensible.

A program is divided into **functions** or **subroutines** where each function has a clearly defined purpose and a defined interface to the other functions in the program. Further, a number of functions are grouped together into larger entity called a **module**, but the principle remains the same, i.e. a grouping of components that carry out specific tasks. Dividing a program into functions and modules is one of the major characteristics of structured programming.

By dividing the whole program using functions, a structured program minimizes the chance that one function will affect another. Structured programming helps the programmer to write an **error free code** and **maintain control** over each function. This makes the development and maintenance of the code **faster** and **efficient**.

Structured programming remained the leading approach for almost two decades. With the emergence of new applications of computers the demand for software arose with many new features such as **GUI (Graphical user interface)**. The complexity of such programs increased multi-fold and this approach started showing new problems.

The problems arose due to the fundamental principle of this paradigm. The whole emphasis is on doing things. Functions do some activity, maybe a complex one, but the emphasis is still on doing. *Data are given a lower status*. For example in banking application, more emphasis is given to the function which collects the correct data in a desired format or the function which processes it by doing some summation, manipulation etc. or a function which displays it in the desired format or creates a report. But you will also agree that the important part is the data itself.

The major drawback with structured programming are its primary components, i.e., *functions and data structures.* But unfortunately *functions and data structures* do not model the real world very well. Basically to model a real world situation data should be given more importance. Therefore, a new approach emerges with which we can express solutions in terms of real world entities and give due importance to data.

Object Oriented programming

The world and its applications are not organized as functions and values separate from one another. The problem solvers do not think about the world in this manner. They always deal with their problems by concentrating on the **objects**, their **characteristics** and **behavior**.

The *world is Object Oriented*, and Object Oriented programming expresses programs in the ways that model how people perceive the world. *Figure 2* shows different real world objects around us which we often use for performing different functions. This shows that problem solving using the objects oriented approach is very close to our real life problem solving techniques.

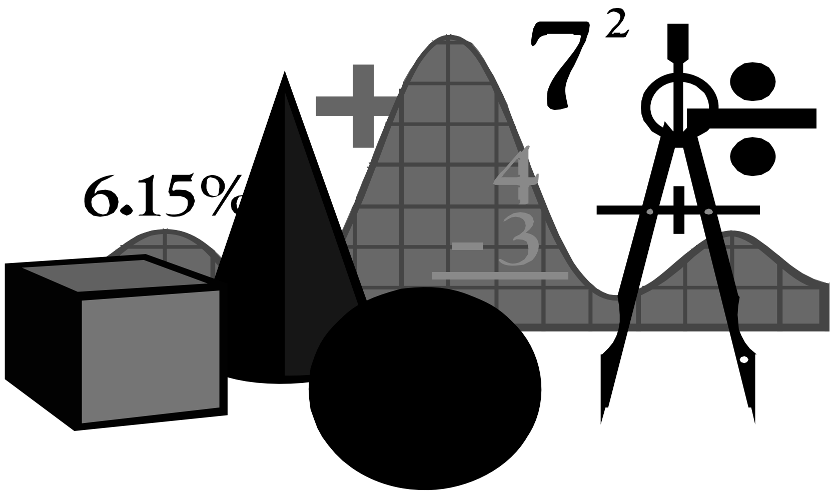


Figure 2: Real world objects

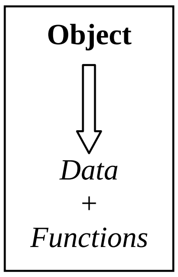
The basic difference in Object Oriented programming **(OOP)** is that the program is organized around the data being operated upon rather than the operations performed. The basic idea behind OOP is to *combine both, data and its functions* that operate on the data into a single unit called **object**. Now in our next section, we will learn about the basic concepts used extensively in the Object Oriented approach.

1.4 BASIC CONCEPTS OF OO APPROACH

Object Oriented methods are favored because many experts agree that Object Oriented techniques are more disciplined than conventional structured techniques. (Martin and Odell 1992)

The main components of Object Oriented technology are '**objects and classes**', '**data abstraction and encapsulation**', '**inheritance**' and '**polymorphism**'. It is very important for you to understand these concepts. Further, in this unit you can find the details of these concepts.

Objects



Let's start with **"Object".** The first thing that we should do in the Object Oriented approach is to start thinking in terms of Objects. The problem to be solved is divided into objects. Start analyzing the problem in terms of objects and the nature of communication between them. Program object should be chosen such that they match closely with real-world objects. Let's start creating objects using real-life things, for example, the **dog.** You can create an object representing a dog, It would have data like **How hungry is it**? **How happy is it**? **Where is it**? Now think what are the different functions you can perform on a dog, like **eat, bark, run** and **dig.** Similarly, the following can be treated as objects in different programming problems:

Fig. 3: Dog object and its behavior

- Employees in a payroll system
- Customers and accounts in a banking system
- Salesman, products, customers in a sales tracking system
- Data structures like linked lists, stacks, etc.
- Hardware devices like magnetic tape drive, keyboard, printer etc.
- GUI elements like windows, menus, events, etc. in any window-based application.

Each object contains ***data and the functions*** that operate on the data. Objects can interact without having to know details of each other's data or functions. It is sufficient to know the type of message accepted and the type of response returned by the object. *For example*, in the banking system, customer object may send a message named as **check balance** to the account object to get the response, i.e. bank balance. An Object Oriented system can be considered as **network of cooperating objects** which interact by sending messages to each other. Let's see in the *Figure 4*, how objects interact by sending messages to one another.

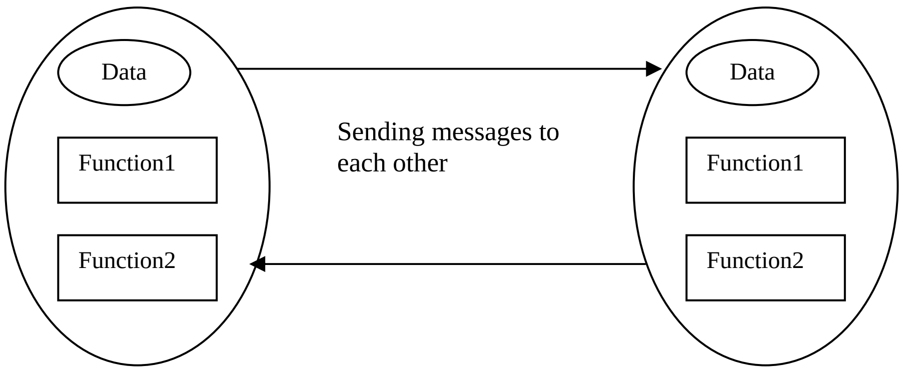


Figure 4: Message Passing

Classes

Objects of the similar type can be grouped together to form a class. Can you tell to which class **dog** belongs? Yes, of course, it belongs to the **animal** class. Now, let us concentrate on the creation of objects. This can be easily answered if we look at the way of creating any variable in common programming languages. Almost all computer languages have **built-in data types,** for example integer, character, real, boolean, etc. One can declare as many variables of any built-in type as needed in any problem solution. In the similar way one can define many objects of the same class. You can take a class as a type created by a programmer.

A class serves as a **plan or template**. The programmer has to specify the entire set of data and functions for various operations on the data for an object as a user-defined type in the form of a class. In other words, **the programmer defines the object format and behavior by defining a class**. The compiler of that language does not know about this user-defined data type. The programmer has to define the data and functionality associated with it by designing a class.

Finally, defining the class doesn't create an object just as the existence of a built-in type integer doesn't create any variable. Once the class has been defined, you can create any number of objects belonging to that class.

A class is thus a *collection of objects of similar type*. For example, in a collection of potatoes each individual potato is an object and belongs to the class potato. Similarly, each individual car running on the road is an object, Collectively these cars are known as cars.

Data abstraction and encapsulation

The wrapping up of data and functions into a single unit is known as **encapsulation**. This is one of the strong features of the object oriented approach. The data is not directly accessible to the outside world and only the functions, which are wrapped in the class, can access it. Functions are accessible to the outside world. These functions provide the interface to access data. If one wants to modify the data of an object, s/he should know exactly what functions are available to interact with it. This insulation of the data from direct access by the program is known as **data hiding**.

Abstraction refers to the act of representing essential features without including the background details to distinguish objects/ functions from other objects/functions. In case of structured programming, **functional abstraction** was provided by telling, ***which task is performed by function*** **and** ***hiding how that task is performed***. A step further, in the Object Oriented approach, classes use the concept of **data abstraction**. With data abstraction, data structures can be used without having to be concerned about the exact details of implementation. As in case of built-in data types like integer, floating point, etc. The programmer only knows about the various operations which can be performed on these data types, but how these operations are carried out by the hardware or software is hidden from the programmer. Similarly in Object Oriented approach, classes act as ***abstract data types***. Classes are defined as a set of attributes and functions to operate on these attributes. They encapsulate all the essential properties of the objects that are to be created.

Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class in the hierarchy. For example, the scooter is a type of the class two-wheelers, which is again a type of (or kind of) the class motor vehicles. As shown in the *Figure 5* the principle behind it is that the derived class shares common characteristics with the class from which it is derived.

New classes can be built from the existing classes. It means that we can add additional features to an existing class without modifying it. The new class is referred as **derived class** or **sub class** and the original class is known as **base class** or **super class**. Therefore, the concept of inheritance provides the idea of **reusability.** This inheritance mechanism allows the programmer to reuse a class that is made almost, but not exactly, similar to the required one by adding a few more features to it.

As shown in *Figure 5*, three classes have been derived from one base class. Feature A and Feature B of the base class are inherited in all the three derived classes. Also, each derived class has added its own features according to the requirement. Therefore, new classes use the concept of reusability and extend their functionality.

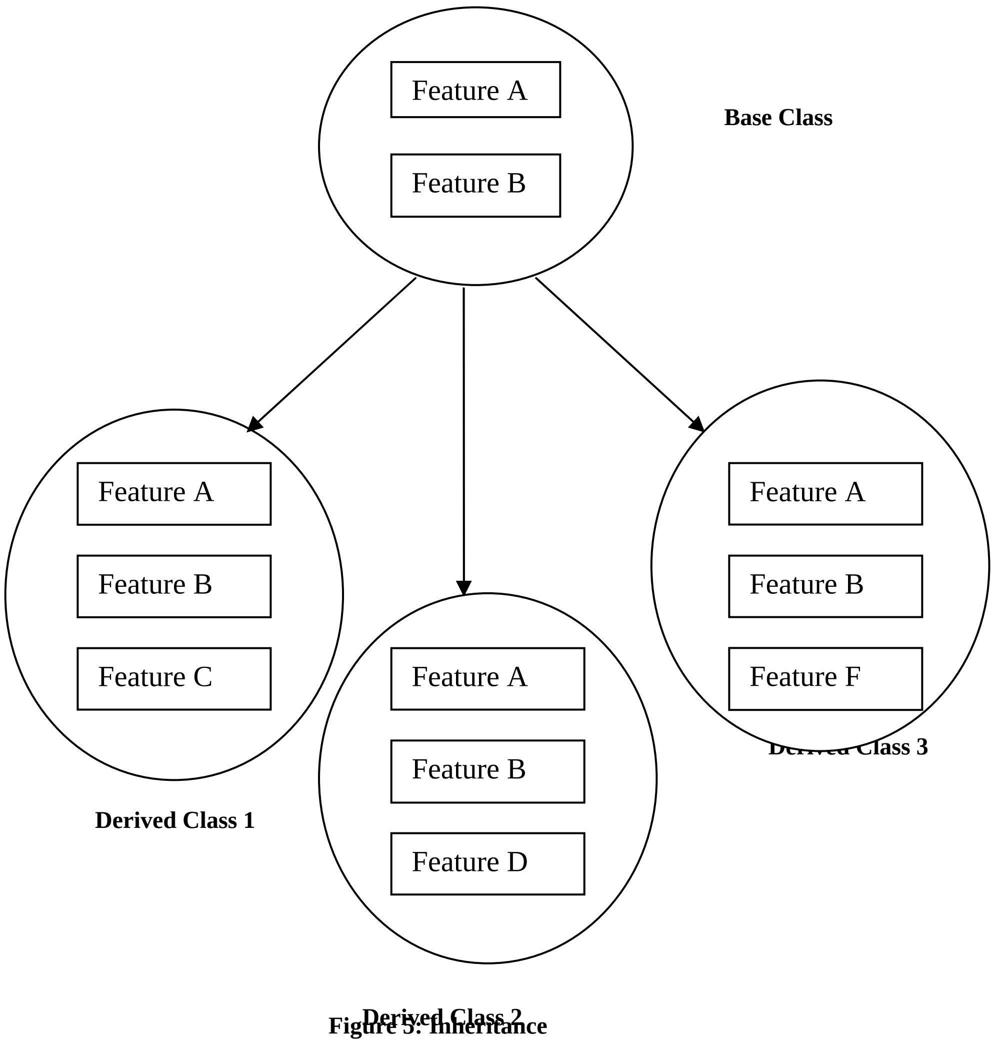


Figure 5: Inheritance

Polymorphism

Polymorphism means **the ability to take more than one form of the same property.** For example, consider an addition operation. It shows a different behavior in different types of data. For two numbers, it will generate a sum. The numbers may integers or float. Thus the addition for integers is different from the addition to floats.

An example is shown in *Figure 6,* where single function name, i.e. **draw** can be used to draw *different shapes*. The name is the same in all the classes but the functionality differs. This is known as function overriding, which is a type of polymorphism. We will discuss it in detail in our next unit.

In our example, we also used a function **"area"** which was inherited by all the three derived classes, i.e. **triangle**, **circle** and **rectangle**. But in the cases of the circle and the triangle, we override the function area because the data types and number of parameters varies.

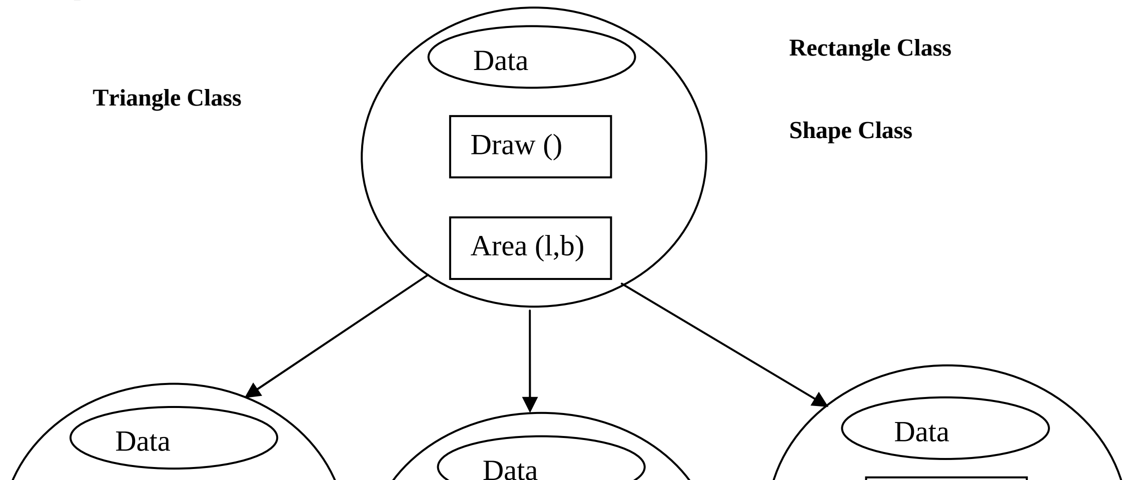


Figure 6: Polymorphism

Check Your Progress 1

1) What do you understand by structured programming?

…………………………………………………………………………………………

……..………………………………………………………………………………………

…

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

………

2) What is the basic idea of Object Oriented approach?

…………………………………………………………………………………………

……..………………………………………………………………………………………

…

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

………

3) Differentiate between Data abstraction and data hiding.

…………………………………………………………………………………………

……..………………………………………………………………………………………

…

…………………………………………………………………………………………

…………………………………………………………………………………………

……

4) Differentiate between Inheritance and polymorphism.

…………………………………………………………………………………

……..…………………………………………………………………………
…

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………
………

1.5 COMPARISON OF OBJECT ORIENTED AND PROCEDURE-ORIENTED APPROACHES

Procedure-oriented approach

A program in a procedural language is *a list of instructions*. Each statement in the language tells the computer to do something. Get some input, do some computational task and finally display the output. This computational task is a function or procedure.

For small programs no other organizing principle is required. The programmer creates the list of instructions, and the computer carries them out. The primary focus is on functions. The program structure can be viewed as shown in *Figure 7*.

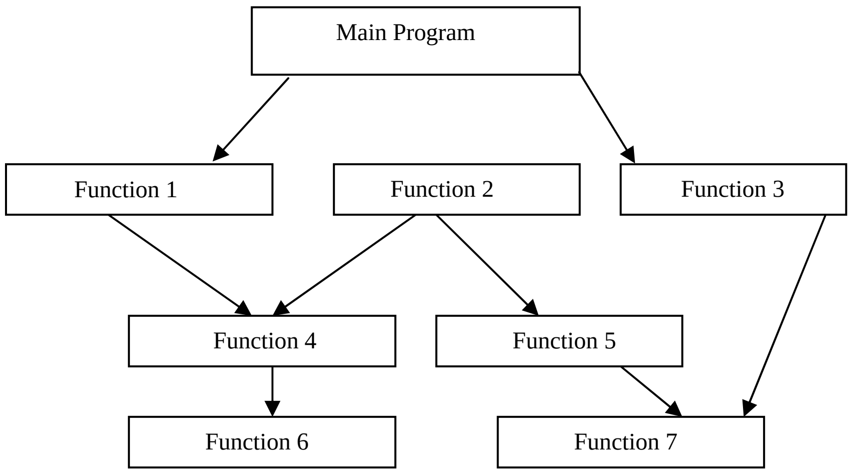


Figure 7: Procedure-Oriented Approach

All conventional programming languages like **Pascal, C, FORTRAN,** and **COBOL** used to model programs based on the procedure re-oriented at approach. In the procedure-oriented approach, the problem is divided into **subprograms** or **modules**. Then functions are defined for each subprogram. Each function can have its own **data** and **logic**. Information is passed between functions using parameters and global variables. Functions can have local variables that cannot be accessed outside the function.

The Programmer starts with thinking *"What do we have to do to solve this problem?* and then s/he does it. Typically it starts with a pseudo code, a flow chart or a **data flow diagram** and continuously refines the design into code. The concentration is more on development of functions. Procedural languages have certain properties, which give rise to some difficulties. The first and the foremost problem is the manner in which functions access global variables. Many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data. The relationship of data and functions in procedure-oriented approach can be viewed as shown in *Figure 8*.

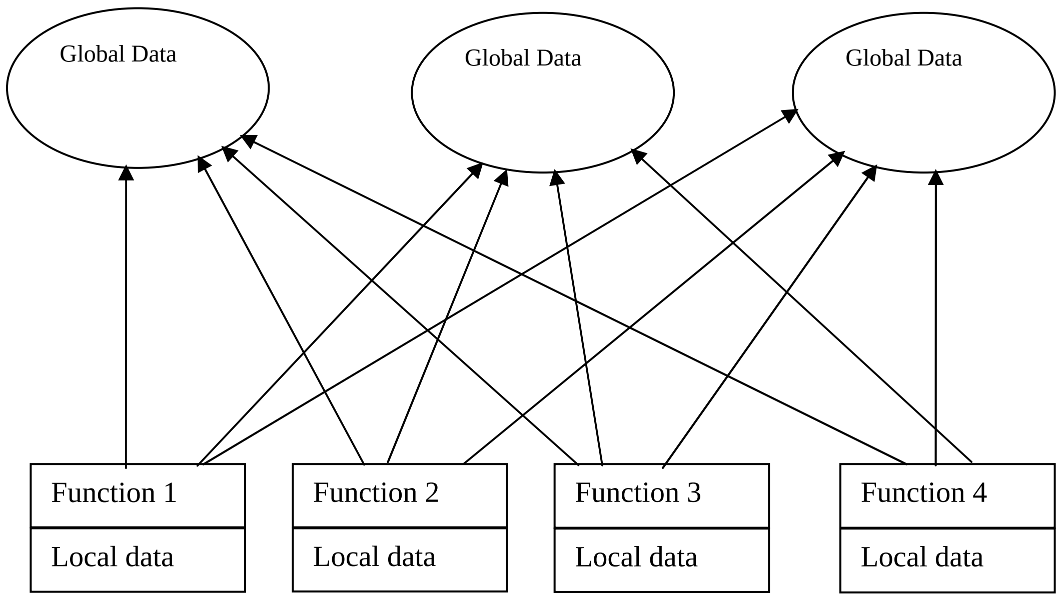


Figure 8: Functions and Global Variables

The following are the two major problems due to global variables:

- **Uncontrolled modification:** Due to these global variables, the risk of unwanted access and modification increases greatly. Any time global variables are available to any function, even to the functions which do not need it. That function may inadvertently access and modify the values without in any way violating the rules of the language. There is no way that this accidental misuse can be avoided and also no method by which we can enforce that the data definition and its use be confined to the functions which actually need them.

- **Problem of enhancement and maintainability:** In a large program it is difficult to keep track of what data is used by which function. If a programmer wants to add some new feature or to make some modification to the existing feature, there are chances that the new code he would add could modify some unprotected global data. If this modification is inappropriate, some part of the program, which is working fine, may stop to do so. Revising this may in turn may need some more modification. This may lead to an opportunity for errors to creep in.

The next issue is the manner in which procedural languages specify the user-defined data type. In these languages, the data type specifies the structure and the operations that can be performed on the variables of that type. For example, in C language, a built-in data type like *int* specifies the structure and the operations applicable to a variable of type *int*. But in these languages, it is not applicable for user-defined. In C language, a data type specifies only the structure of the variables of that type. The operations that are to be performed on a variable of this type are left unspecified. As a result, it may not be clear? What manner the variables are to be operated upon? It is left to the user program to decide the type of operations that are to be performed upon variables of that type.

For example we can specify the employee record using ***struct*** and using it. An array of employees is defined and it is possible to access employee information by accessing each element of the array. To have a richer specification, we can associate operations like ‘compute salary’, ‘print leave details’ etc. In the procedure; oriented approach, the programmer has to define these operations outside the structure.

The next issue is of **reusability**. Since procedural languages are strongly typed, their functions are highly dependent on the type of variables that are being used. This property hampers reusability. *For example,* if a program for sorting were written for integers, it would not be able to sort real numbers. For that a new program has to be written.

Finally the serious drawback of the procedural approach is that it *does not model the real world problems very well*. The emphasis is more on functions that represents the action or activity and does not really correspond to the elements of the problem.

The major characteristics of the procedure-oriented approach are:

- More emphasis is on doing things.
- It is based on the problem at hand. Sequence or procedure or functionality is paramount.
- Most of the functions share global data which may lead to the following problems:
  - It will not produce software that is easy to maintain,
  - The risk of unwanted access and modification increases.
- It will not result in reusable software.
- It employs top-down approach in program design.
- Works well in small systems.

Object Oriented approach

The major factor, which leads to the development of this new approach i.e, Object Oriented approach is to resolve many problems encountered earlier in the procedural approach.

In this approach, we decompose a problem into a number of entities called objects and then build data and functions around these entities. The notion of “Object” comes into the picture. *‘A collection of data and its operations is referred to as an object’*. Data is a vital element in the program development. Data is local to an object. This is encapsulated within an object and is not accessible directly from outside the object. These objects know how to interact with another object through the interface (a set of operations). The organization of data and functions in Object Oriented programs is shown in *Figure 9* given below.

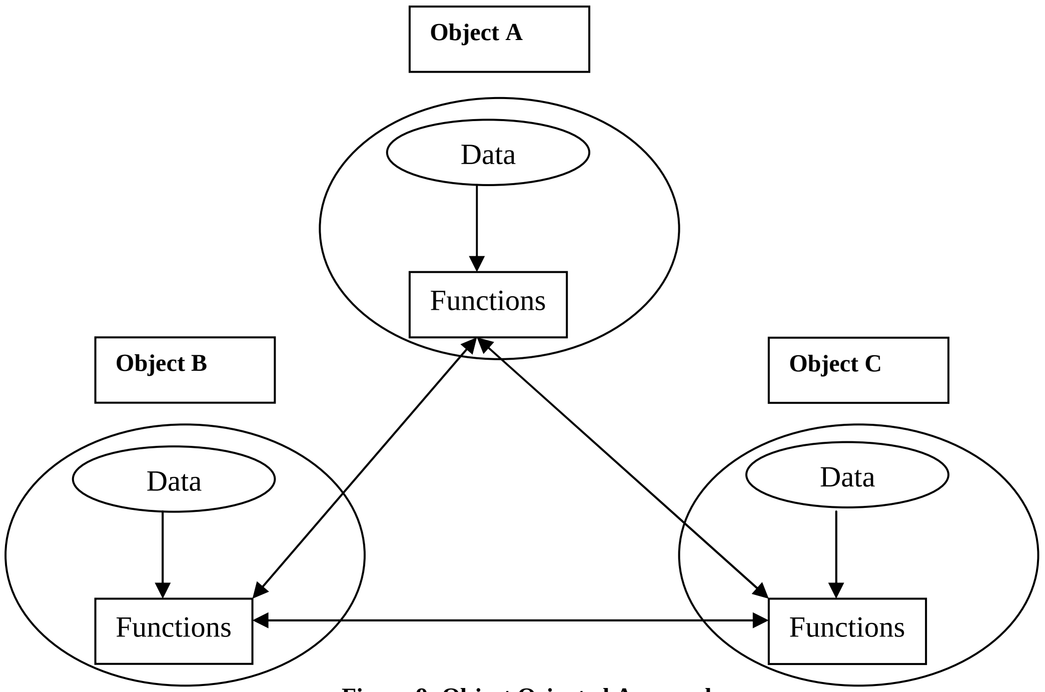


Figure 9: Object Oriented Approach

The salient features of Object Oriented programming are:

- More emphasis is on data rather than procedure.
- Programs are modularized into entities called *objects*.
- Data structures methods characterize the objects of the problem.
- Since the data is not global, there is no question of any operations other than those defined within the object, accessing the data. Therefore, there is no scope of accidental modification of data.
- It is easier to maintain programs. The manner in which an object implements its operations is internal to it. Therefore, any change within the object would not affect external objects. Therefore, systems built using objects are resilient to change.
- Object reusability, which can save many human hours of effort, is possible. An application developer can use objects like ‘array’, ‘list’, ‘windows’, ‘menus’, ‘event’ and many other components, which were developed by other programmers, in her program and thus reduce program development time.
- It employs bottom-up approach in program design.

1.6 BENEFITS OF OOPS

OOP offers several benefits to both the program developer and the user. The new technology provides greater programmer productivity, better quality of software and lesser maintenance cost. The major benefits are:

- **Ease in division of job:** Since it is possible to map objects of the problem domain to those objects in the program, the work can be easily partitioned based on objects.
- **Reduce complexity:** Software complexity can be easily managed.
- **Provide extensibility:** Object Oriented systems can be easily upgraded from small to large system.
- **Eliminate redundancy:** Through inheritance we can eliminate redundant code and extend the use of existing classes.
- **Saves development time and increases productivity:** Instead of writing code from scratch, solutions can be built by using standard working modules.
- **Allows building secure programs:** Data hiding principle helps programmer to build secure programs that cannot be accessed by code in other parts of the program.
- **Allows designing simpler interfaces:** Message passing techniques between objects allows making simpler interface descriptions with external systems.

Check Your Progress 2

1) State True or False.

   a) In the procedure-oriented approach, all data are shared by all functions. ☐

   b) One of the major characteristics of OOP is the division of programs into objects that represent real-world entities. ☐

   c) Object Oriented programming language permit reusability of the existing code. ☐

   d) Data is given a second-class status in procedural programming approach. ☐

   e) OOP languages permit functional as well as data abstraction ☐

2) Does procedure oriented language support the concept of class?

…………………………………………………………………………………

……..………………………………………………………………………… …

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

………

3) Give the reason of accessing data of a class through its functions only.

…………………………………………………………………………………

……..………………………………………………………………………… …

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

………

1.7 INTRODUCTION TO COMMON OO LANGUAGE

The language should support several of the OO Concepts to claim that they are object oriented. Depending on the features they support, *they can be classified into the following two categories:*

- Object-based programming languages,
- Object Oriented programming languages.

Object-based programming is the style of programming that primarily supports encapsulation and object identity. *Major features that are required for object-based programming are:*

- Data encapsulation
- Data hiding and access mechanisms
- Automatic initialization and clear-up objects
- Polymorphism.

Languages that support programming with objects are said to be object-based programming languages. They do not support inheritance and dynamic binding. Ada is a typical object-based programming language.

We have already introduced to you the concept of data encapsulation, data hiding, polymorphism and inheritance in previous sections of this unit. The access mechanism is implemented through various access specifiers such as public, protected, and private. The function of these specifiers is to decide the scope of a particular member within software. This provides the convenient way for the programmer to decide about the **accessibility** of a member for others.

The other issue is automatic initialization and clear-up objects. Automatic initialization means to give initial values to various variables that denote the state of the object at the time of object creation. This is implemented using ***constructors***. *Finally* when there is no use of the object, we can destroy it. At that time all the resources used by it must be cleared-up. This is implemented using ***destructors***. We will discuss about this in due course.

Let us discuss the concept of **dynamic binding**. Binding refers to the linking of objects to their properties or method call to the code to be executed in response to a call. It can be done at *compile time or run time*. Dynamic binding refers to the linking done at the run (executor) time. We will study this in detail in our coming units of the next blocks.

Object Oriented programming incorporates all of object-based programming features along with additional features such as *inheritance* and *dynamic binding*. Examples of Object Oriented languages are **C++, Smalltalk, object-Pascal,** and **Java**.

Use of a particular language depends on characteristics and requirements of an application, organizational impact of the choice and reuse of the existing programs. *Java is becoming the most widely used general purpose OOP language in the computer industry today.*

1.8 APPLICATIONS OF OOPs

Applications of OOPs are gaining importance. There is a lot of excitement and interest among software developers in using OOPs. The richness of the OOP environment will enable the software industry to improve not only the quality of the software systems but also its **productivity**. Object Oriented technology is certainly changing the way software engineers think, analyze, design and implement systems.

The most popular application using Object Oriented programming is the interface designing for window base systems. Real systems are more complex and contain many objects with a large number of attributes and methods of complex nature. OOP is useful in such cases because it can simplify a complex problem. The application area of OOP includes:

- Object Oriented databases
- Embedded systems
- Simulation and modeling
- Neural networks
- Decision support systems
- Office automation systems
- AI and expert systems
- CAD/CAM systems
- Internet solutions.

Check Your Progress 3

1) State True or False

   a) Protecting data from access by unauthorized functions is called data hiding. ☐

   b) Wrapping up of data of different types and functions into a unit is known as encapsulation. ☐

   c) Polymorphism can be used in implementing inheritance. ☐

   d) A Class permits us to build user-defined data types. ☐

   e) Object Oriented approach cannot be used to create databases. ☐

2) Explain the advantage of dynamic binding.

…………………………………………………………………………………………
…

…………………………………………………………………………………………

…………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

…………

3) Differentiate between object based and object oriented programming languages

………………………………………………………………………………………

…………..………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

.

1.9 SUMMARY

OOP is a new way of organizing and developing programs. It eliminates many pitfalls of the conventional programming approach. OOP programs are organized around objects, which contain data and functions that operate on that data. A class is a template for a number of objects. The object is an instance of a class. The major features of OOP are data abstraction, data encapsulation, inheritance and polymorphism. This new methodology increases programmer productivity, delivers better quality of software and lessens maintenance cost. Languages that support several OOP concepts include C++, Smalltalk, Object Pascal and Java.

1.10 SOLUTIONS/ANSWERS

Check Your Progress 1

1) In structured programming, a program is divided into functions or modules and each module has a clearly defined purpose and a defined interface to the other functions in the program. Dividing a program into functions and modules is one of the major characteristics of structured programming.

   Here we are least bothered about the data of the problem provided. Our main objective is to achieve control of the execution of program correctly.

2) In Object Oriented programming (OOP), the program is organized around the data being operated upon rather than the operations performed. The basic idea behind OOP is to combine both, data and its functions that operate on the data into a single unit called object.

3) In data abstraction, data structures are used without having to be concerned about the exact details of implementation.

   This insulation of the data from direct access by the other elements of the program is known as data hiding. It is achieved through classes in OOPs.

4) Inheritance is the process by which objects of one class acquire the properties of objects of another class in the hierarchy. By using inheritance, new classes can be built from the existing old classes. It means that we can add additional features to an existing class without modifying it. This inheritance mechanism allows the programmer to reuse a class that is almost, but not exactly, similar to the required one by adding a few more features to it.

Polymorphism means the ability to take more than one form with the same name. Using polymorphism we can have more than one function with the same name but with different functionalities.

Check Your Progress 2

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

2) Yes procedural languages also support the concept of class, for example, type (data type of the language) is a class and is supported by procedural languages. You know C language support several data types. But procedural languages don't support the user-defined class that has data and functions together.

3) Accessing data of a class through its functions is in basic philosophy of object orientation. If data is not having restrictive access and open to all the principle of data hiding is violated and emphasis on data get reduced.

Check Your Progress 3

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

2) It gives option of run-time selection of methods on the basis of current input during execution of program. Dynamic binding allows new objects and code to be interfaced with or added to a system without affecting existing code.

3) Object based languages support the notion of objects. Object Oriented languages support the concept of class and permit inheritance between classes.