5.5 APPROACHES TO SOFTWARE DESIGN

There are two fundamentally different approaches to software design that are in use today — function-oriented design, and object-oriented design. Though these two design approaches are radically different, they are complementary rather than competing techniques. The object-oriented approach is a relatively newer technology and is still evolving. For development of large programs, the object-oriented approach is becoming increasingly popular due to certain advantages that it offers. On the other hand, function-oriented designing is a mature technology and has a large following. Salient features of these two approaches are discussed in subsections 5.5.1 and 5.5.2 respectively.

5.5.1 Function-Oriented Design

The following are the salient features of the function-oriented design approach:

Top-down decomposition

A system, to start with, is viewed as a black box that provides certain services (also known as **high-level functions**) to the users of the system.

In top-down decomposition, starting at a high-level view of the system, each high-level function is successively refined into more detailed functions.

For example, consider a function `create-new-library member` which essentially creates the record for a new member, assigns a unique membership number to him, and prints a bill towards his membership charge. This high-level function may be refined into the following subfunctions:

- `assign-membership-number`
- `create-member-record`
- `print-bill`

Each of these subfunctions, may be split into more detailed subfunctions, and so on.

Centralized system state

The system state can be defined as the values of certain data items that determine the response of the system to a user action or external event. For example, the set of books (i.e. whether borrowed by different users or available for issue) determines the state of a library automation system. Such data in procedural programs usually have global scope and are shared by many modules.

The system state is centralized and shared among different functions.

For example, in the library management system, several functions such as the following share data such as `member-records` for reference and updation:

- `create-new-member`
- `delete-member`
- `update-member-record`

A large number of function-oriented design approaches have been proposed in the past. A few of the well-established function-oriented design approaches are the following:

- Structured design by Constantine and Yourdon, [1979]
- Jackson's structured design by Jackson [1975]
- Warnier-Orr methodology [1977, 1981]
- Step-wise refinement by Wirth [1971]
- Hatley and Pirbhai's methodology [1987]

5.5.2 Object-Oriented Design

In the object-oriented design (OOD) approach, a system is viewed as being made up of a collection of objects (i.e. entities). Each object is associated with a set of functions that are called its methods. Each object contains its own data and is responsible for managing it. The data internal to an object cannot be accessed directly by other objects and only through invocation the methods of the object. The system state is decentralized since there

is no globally shared data in the system and data is stored in each object. For example, in a Library Automation Software, each library member may be a separate object with its own data and functions to operate on the stored data. The methods defined for one object cannot directly refer to or change the data of other objects.

The object-oriented design paradigm makes extensive use of the principles of abstraction and decomposition as explained below. Objects decompose a system into functionally independent modules. Objects can also be considered as instances of abstract data types (ADTs). The ADT concept did not originate from the object-oriented approach. In fact, ADT concept was extensively used in the ADA programming language introduced in the 1970s. ADT is an important concept that forms an important pillar of object-orientation. Let us now discuss the important concepts behind and ADT. There are, in fact, three important concepts associated with an ADT: data abstraction, data structure, data type. We discuss these in the following:

Data abstraction

The principle of data abstraction implies that how data is exactly stored is abstracted away. This means that any entity external to the object (that is, an instance of an ADT) would have no knowledge about how data is exactly stored, organized, and manipulated inside the object. The entities external to the object can access the data internal to an object only by calling certain well-defined methods supported by the object. Consider an ADT such as a `stack`. The data of a stack object may internally be stored in an array, a linearly linked list, or a bidirectional linked list. The external entities have no knowledge of this and can access data of a stack object only through the supported operations such as `push` and `pop`.

Data structure

A data structure is constructed from a collection of primitive data items. Just as a civil engineer builds a large civil engineering structure using primitive building materials such as bricks, iron rods, and cement; a programmer can construct a data structure as an organized collection of primitive data items such as integer, floating point numbers, characters, etc.

Data type

A type is a programming language terminology that refers to anything that can be instantiated. For example, `int`, `float`, `char`, etc. are the basic data types supported by C programming language. Thus, we can say that ADTs are user defined data types.

In object-orientation, classes are ADTs. But, what is the advantage of developing an application using ADTs? Let us examine the three main advantages of using ADTs in programs:

- The data of objects are encapsulated within the methods. The encapsulation principle is also known as data hiding. The encapsulation principle requires that data can be accessed and manipulated only through the methods supported by the object and not directly. This localizes the errors. The reason for this is as follows. No program element is allowed to change a data, except through invocation of one of the methods. So, any error can easily be traced to the code segment changing the value. That is, the method that changes a data item, making it erroneous can be easily identified.
- An ADT based design displays high cohesion and low coupling. Therefore, object-oriented designs are highly modular.
- Since the principle of abstraction is used, it makes the design solution easily understandable and helps to manage complexity.

Similar objects constitute a class. In other words, each object is a member of some class. Classes may inherit features from a super class. Conceptually, objects communicate by message passing. Objects have their own internal data. Thus, an object may exist in different states depending the values of the internal data. In different states, an object may behave differently. We shall elaborate these concepts in Chapter 7 and subsequently we discuss an object-oriented design methodology in Chapter 8.