BLOCK II:
OBJECT ORIENTED DESIGN

UNIT 1: INTRODUCTION TO OBJECT ORIENTED DESIGN

Space for Learners:

Unit Structure:

1.1 INTRODUCTION

The object-oriented programming is widely used to solve different kinds of computing problems. The concept of object-oriented programming is to find solution to the problem based on real-life examples. Object-oriented design is used to prepare a plan to solve the software related problem diagrammatically. The whole design is based on objects and classes where objects interact with each other to solve a problem. In this unit, we will learn the object-oriented approaches, the key concepts of object-oriented design and different components of it.

Object-oriented design is a process to design software. The fundamental steps of developing software are: Requirement analysis, designing, coding, testing and maintenance of the software. There are many techniques and tools to develop software. The need for a tool may vary as per the requirement of the user and the change in the technology. Among all other design strategies, object-oriented development techniques are a popular one.

Space for Learners:

1.2 UNIT OBJECTIVES

The main objectives of this unit are:

- To familiarize with the object-oriented programming history
- To know the purpose and benefits of using object-oriented approaches
- To have a clear concept on different concepts related to object-oriented design
- To gather the knowledge on the importance of object-oriented analysis and design
- To know the use of link, association and basics object-oriented design

1.3 BRIEF HISTORY

In this section, we will show the evaluation of object-oriented programming with time:

- The term 'object' and 'oriented' are in discussions at MIT in the early 1960s.
- Alan Kay presented a detailed concept on "Object-Oriented Programming" in 1966-1967.
- In MIT, an ALGOL version named AED-0 was initiated which was capable of establishing a link between the data structures and procedures.
- A programming language name Simula was designed using object-oriented concept in the late 1960s. It was capable of

run on the UNIVAC 1107 computer. Simula introduced the fundamental concepts of OOP like class, object, inheritance and dynamic binding.

- The first version of Smalltalk programming language was built by Alan Kay, Dan Ingalls and Adele Goldberg inthe1970s.
- Inmid-1980s, Brad Cox developed Objective-C which proved to be new beginning in OOP history. In parallel to this, Bjarne Stroustrup created the Object-Oriented C++.
- Grady Booch proposed the design concept in a programming language in a paper titled Object Oriented Design.
- In the 1990s, the popularity of using object-oriented programming began using the languages like C++, Visual FoxPro 3.0 etc.
- Eventually, object-oriented concepts were added to many languages like ADA, Fortran, Pascal, Basic, COBOL etc.
- In present days, languages like Python, Ruby, JAVA by Sun Microsystems, C#, Visual Basic.NET have completely emerged as object-oriented programming languages. And these languages are widely used in various fields.

Space for Learners:

1.4 OBJECT-ORIENTED SOFTWARE DEVELOPMENT METHODOLOGY

This methodology helps the programmer to work based on object-oriented concepts to solve the problems. Object-oriented software development methodology includes the concept of requirement analysis, object-oriented design and implementation of the system. Unlike the traditional software development methods, it focuses on developing software using objects. These objects can be easily created, removed, modified and reused. Objects are also able to communicate between themselves.

1.5 OBJECT-ORIENTED APPROACHES

The object-oriented paradigm focuses on solving a problem using both theoretical and conceptual knowledge. Here, a system is assumed as a collection of various entities which are able to work and interact together to fulfil certain objectives. These entities may be physical (like animal, flower, person etc.) or maybe abstract concepts (like files, function etc.). In object-oriented analysis, these entities are known as objects.

In an object-oriented approach, these objects consist of data and procedures. The main aim is to propose a design to improve the productivity and quality of the system. The whole object-oriented programming approach can be partitioned into three main phases. The first phase is Object-oriented Analysis (OOA) phase, where the requirements are analysed and the problem domain is identified in terms of objects. The Object-oriented design phase works to design a solution domain based on the objects identified in the previous phase. Object-oriented programming works on the implementation of the system using object-oriented concepts.

The benefits of object-oriented programming approaches are:

- Object-oriented system model works in more organized way than other traditional approaches.
- Object-oriented system can be designed and code easily.
- The maintenance cost is comparatively low in this approach.
- Object-oriented programming allows the reusability of design and code.
- This programming is more adaptive in nature. Modifications can be done easily as per requirement.
- This programming is reliable and more flexible for the programmers.

1.6 OBJECT-ORIENTED ANALYSIS (OOA)

Object-oriented Analysis is the starting phase of the object-oriented software development procedure. This phase gathers all the requirements to develop the system and identifies all the classes and relations between the classes. Objects are the instances of the class. All the requirements are organized as objects. OOA focuses to create real-world models using the object-oriented view of the real world.

Space for Learners:

STOP TO CONSIDER

Grady Booch coined the term Object-Oriented Analysis as "Object-Oriented Analysis is a method of analysis that examines requirements from the perspective of classes and objects found in the vocabulary of the problem domain".

1.6.1 Steps of Performing Object-Oriented Analysis

The working of OOA is shown in **Figure 1.1**. The key steps of performing **object-oriented analysis** are:

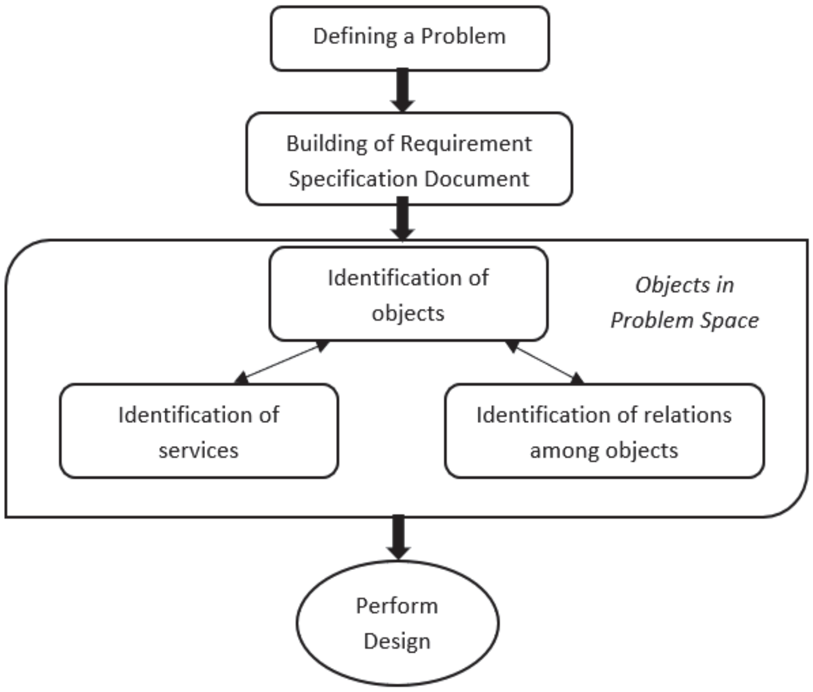
Defining the Problem: The first step in OOA is to analyse the problem for better understanding. The problem needs to be studied in detail and redefine the problem in engineering, so that a computer-based solution can be prepared stepwise from the problem definition using object-oriented concepts.

Requirement Analysis: The second step is to gather all the user requirements. From this, we can proceed to propose the solution domain in the design phase. A requirement specification document should be prepared based on all user requirements and the software requirement for solving the problem. The specification document should contain what the system actually does, what type of inputs are necessary, what are the outputs that system produces and how processes can be built to generate required outputs.

Identification of Objects: In this step, we need to collect all the objects and list the attributes contained in the object based on the requirement specification document built in the previous step. Objects can be related to real-life entity or can be of an abstract type.

Deciding the services of objects: After the identification of objects, the next step is to identify the services to be provided by each object. Each object is assigned some services to be performed. The strength of the object-oriented paradigm is that once the services are provided to the objects, it is guaranteed to be accomplished by the objects.

Establishing relationship among the objects: Objects need to interact among themselves to perform different services to produce better results. This works on identifying the relationships among the objects so that objects can communicate during execution.



Defining a Problem
Building of Requirement Specification Document
Identification of objects
Objects in Problem Space
Identification of services
Identification of relations among objects
Perform Design

Figure 1.1: Working of object-oriented analysis

1.7 OBJECT-ORIENTED DESIGN (OOD)

The classes identified in the OOA phase are designed in the design phase. This phase also helps to build the user interfaces. As per the requirement, more numbers of classes and objects can be added in the design phase. Object-oriented design helps to build the solution domain by identifying the classes, relationship between them, identifying the constraints and designing the user interfaces. The outputs of object-oriented analysis are taken as input to the object-oriented design.

Object-oriented design follows the concept of **decomposition**. Here, decomposition means to divide the whole system into the hierarchy of various components. Each component has its own characteristics and functions. These smaller components are known as subsystems. OOD establishes communication between the subsystems using corresponding objects. The main advantage of decomposing the system in OOD is that all the subsystems are of lesser complexity, so they can be easily understandable and manageable. And the

Space for Learners:

subsystems can be easily modified or replaced depending on the situation without affecting the other subsystems. In OOD, to show the interaction between two classes and objects, associations and links are used.

Object-oriented design is composed of two main types of design: Object Design and System Design. We will study this detailed design in Unit 3. The models built using OOD are either static models or dynamic models. Based on that OOD can be categorized as of mainly three models:

i) Object Model
ii) State Model or Dynamic model
iii) Functional model

These three models can be represented with different diagrams. Object model can be represented with **class diagram** and **object diagram**. State models can be drawn **using state diagram**. **Data Flow Diagram (DFD)** is generally used to represent functional model. We will learn the functionalities of these three models and to design their respective diagrams using **Unified Modeling language (UML)** in the next unit.

In present days, many of the software designer use another model known as **interaction model** instead of functional model. The state diagrams of dynamic model focus on their respective class. But sometimes it becomes difficult to understand the entire system. Interaction model focuses on the relationship between the classes, how the objects interact with others to display efficient result. To present the interaction model, **Use Case diagrams, Sequence diagrams** and **Activity diagrams** are commonly used.

STOP TO CONSIDER

Grady Booch defined the term OOD as, "Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design".

Space for Learners:

1.7.1 Goals of Object-Oriented Analysis and Design (OOAD)

Both OOA and OOD are together known as **Object-Oriented Analysis and Design (OOAD)** which includes both the requirement analysis, identification of classes and design part of the object-oriented system. The different objectives or goals of object-oriented design and analysis are:

The main objective of performing object-oriented analysis and design is that it helps the programmer to build an efficient solution to complex problems based on the object-oriented concepts.

- Following the object-oriented analysis and design step, software development team management can have a better understanding of the problem domain.
- Graphical representations of the problem make it easier for the team management, stakeholders and the end-user to work together and check whether the software building process is on the right track or not.
- It increases the reusability of the project.
- It is performed at the abstract level which helps to separate the actual programming part from the design.

1.8 OBJECT-ORIENTED PROGRAMMING (OOP)

Object-oriented programming is a pattern of writing programs where data and functions are put together in a skeleton-like structure called class, where data need to be processed and functions need to be performed. And the whole operation is executed using any instance of a class known as an object. There can be multiple classes in a program and a hierarchy of classes can be maintained using the concept of inheritance. Along with this, object-oriented programming follows the concepts of abstraction, encapsulation and polymorphism. This programming technique uses different access specifier to maintain the data hiding property.

Data hiding properties protect the data and member functions of a class from unauthorised access outside the class. The reason behind the popularity of object-oriented programming language is that it is intact with real-world examples and so, has the capability of solving complex problems in a better way.

Space for Learners:

STOP TO CONSIDER

Grady Booch defined the term OOP as, "Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of hierarchy of classes united via inheritance relationships".

1.8.1 Objects

An object can be said as an identity that can be categorized by its behaviour. It is a component consisting of different properties and methods to make data useful. M. R. Blaha and J. R. Rumbaugh defined the term object in their book [2] as, "An object is a concept, abstraction, or thing with the identity that has meaning for an application". Simply, an object is called as an instance of a class. No memory is allocated during the defining of class. Memories are allocated to the objects of a class based on the size of data members (variables, constants etc.) defined inside the class.

Objects can be related with real life entity. For example, if we consider a class name as *flower*, then a single flower of any kind (say *lotus* or *rose*) can be considered as an individual object. Objects can also be of an abstract type which has conceptual existence. For example, if we consider a class name as *examination*, here different subjects like *subject1, subject2* etc. can be objects.

1.8.2 Classes

We have read an object as an instance of a class. So, a class is collection of similar kind of objects which possesses similar characteristics. A class is skeleton of program which consist of data members and the member functions. The term data member refers to the variable or constants declared within a class. The scope of these variables is within this class. The member functions are the procedures or functions defined in the class. The class does not consume any memory. These data members and member functions of the class can only be accessed through the objects of that class. So, the objects consist of all the characteristics defined in the class.

All the objects of a class have the same number of attributes and properties. By grouping set of similar objects relating to a class, the concept of abstraction is introduced in the object-oriented programming. As in the previous example, all the properties of a flower are stored into the class *flower*. Any objects of class *flower* possess similar kind of properties.

Later in this unit and next unit, we will learn how to draw class diagram and object diagram in OOD.

Space for Learners:

CHECK YOUR PROGRESS - I

Fill in the blanks:

1. ____________________ is collection of similar kind of objects which possesses similar characteristics.
2. Instance of a class known as an _________________

   ____________ properties protect the data and member functions of a class from unauthorised access outside the class
3. _____________________ design is a process to design software.

1.9 DIFFERENT CONCEPTS OF OBJECT-ORIENTED DESIGN (OOD)

In this section, we will study important concepts related to object-oriented design and object-oriented programming.

1.9.1 Abstraction

Abstraction is a process of hiding the background detail from the user. It is one fundamental method to deal with complexities. We use different machines like any electronic device, electrical device, mechanical device etc. for their daily use. These devices workin a complex way as it looks from the outside. But to gather full knowledge on how exactly a device performs before using a device would be tougher for humans. People can use a device if they know how exactly the device operates or what will be the outcome based

Space for Learners:

on the given input. There is the concept of abstraction lying on. This concept allows hiding all the working procedures, functionalities that are not necessary for using a particular device. Let us take an example of an electric fan. We know that as we switch on the fan, it will start rotating and help us to cool within a certain period of time. When we switched off the fan, it stops. This information is enough for us to have the benefit of an electric fan. But behind this, many processes have to take place, conversion of energies has been going on. Abstraction is the concept to separating all the complex unnecessary information from the normal users.

In object-oriented programming also, the concept of abstraction is playing the same role. Abstraction focuses on to separate the important purpose from the unimportant aspects. For a particular thing, many different abstractions also can be possible based on the purpose of their use. The characteristics of a good model to identify the important information of a problem and removes the other.

STOP TO CONSIDER

Grady Booch defined the abstraction as "An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer".

1.9.2 Encapsulation

Encapsulation is also based on the concept of information hiding. Abstraction deals with the characteristics of the object but encapsulation focuses on the internal implementation for which characteristics of the objects are raised. Encapsulation hides the detailed information from the other objects. For this, the modification in any part of a code does not affect the whole program.

Encapsulation is a process of binding data with related methods and it prevents its accessibility from other objects. The programs must be encapsulated in the proper way. It is mainly used to hide the internal detail from the outside objects.

STOP TO CONSIDER

Grady Booch coined encapsulation as, "Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behaviour; encapsulation serves to separate the contractual interface of an abstraction and its implementation".

Space for Learners:

1.9.3 Inheritance/ Hierarchy of Class

Inheritance is one of the key properties of object-oriented programming. Inheritance allows a class or object to inherit the characteristics of other classes or objects. The hierarchy of the classes states that one class can access the data and member functions of another class.

The class from which the properties can be inherited is known as base class or super class or parent. The class which inherits the properties of the parent class is known as derived class or subclass or child class. In C++, a child class can inherit the properties of multiple parent classes whereas a parent class can have multiple child classes. C++ supports different types of inheritance like single inheritance, multiple inheritance, multilevel inheritance, hierarchical inheritance and hybrid inheritance. We have already studied the functionalities of different types of inheritance in this paper.

STOP TO CONSIDER

Grady Booch defined the term hierarchy as, "Hierarchy is a ranking or ordering of abstractions".

Here, we try to learn the organization of class hierarchies with an example. Let us take the example of different types of insurance policy. A person can obtain insurance from any brand or company. First the type of insurance is selected. Insurance can be of many types like health insurance, life insurance, motor vehicle insurance. All the insurance may have some common properties like brand name, number of years, policy number etc. But the features, procedures and benefits vary depending on type of insurance. So, in figure 1.2, the main or parent class is termed as *Insurance_Policy* which may contain all the common properties. Based on the type, it is divided into three child or subclasses, viz. *Health_Insurance*, *Life_Insurance*, *Motor_Vehicle_Insurance*. The second hierarchy level classes contain their features. Further, these child classes also can be decomposed into more subclasses. For example, the feature of health insurance may vary based on whether it is for self or for family. Like *Life_Insurance* class and *Motor_Vehicle_Insurance* class also can be divided. Thus, in OOP multiple classes can be organized in the level of hierarchy. Objects of lower-level classes can access the properties of their parent classes, whereas parent classes cannot access the properties of their child classes.

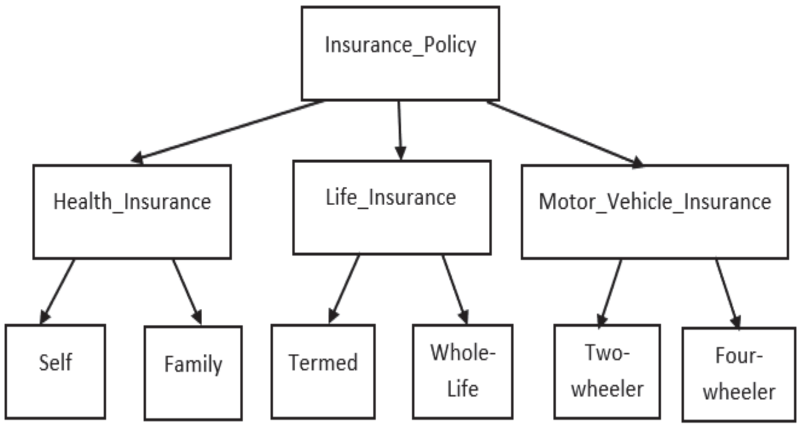


Figure 1.2: Hierarchy of classes types of insurance policy to understand inheritance

1.9.4 Modularity

Modularity is the concept of splitting a program into single components with aim to reduce complexity of the program. These different components are known as modules. It can be said as the degree to which it can be separated. It utilizes the concepts of abstraction and encapsulation in the form of modules.

STOP TO CONSIDER

Grady Booch defined the term modularity as, "Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules".

1.9.5 Polymorphism

The term polymorphism describes the concept that is an object can obtain many forms depending on the situation. In object-oriented programming also, polymorphism works in similar way. It allows a common external interface to perform two or more operations in a different way based on what they are operating. Let us take an example, suppose a plus sign (+) is used in such a way that when it is placed in between two numbers in an operation, it shows the sum of those two numbers as result. Again, if it is placed between two strings and the operation is performed it gives the concatenated string by combining those two input strings. So, in that case, the plus sign is capable of performing the operation in different ways based on the input variable. We can perform this type of operation using an operator or function in OOP.

CHECK YOUR PROGRESS - II

State True or False:

5. Abstraction is a process of hiding the background details from the user.
6. In binary relationship, connection is established among the objects of same class.
7. In unary relationship the connection is established between objects of two classes.
8. A single object of one class may build relationship with more than one object of other class is known as one-to-many relationship.

1.10 GENERALIZATION AND SPECIALIZATION

The concept of generalization and specialization is related to hierarchy of classes. It is the relationship between a parent class and one or more child classes. The classes are organized by the similarities and differences between the classes and their properties. Generalization is a process to create a class of higher-level hierarchy known as parent class by combining all the common properties of the child classes. The child classes have access to those common properties. With this, child classes may have some other properties.

Specialization is nothing but just the reverse of generalization. Specialization is a process of creating new specialized child classes from the parent class. These specialized classes hold the distinguish properties related to the parent class. It is used when a set of properties (attributes or methods) can be applied to some of the objects of a class, then a new child class is created which holds those special properties. Here, the higher-level class is split into lower-level classes.

Let us try to understand the concept of generalization and specialization with a single example shown in **Figure 1.3**. If we follow the concept of generalization, then two classes Polygon and Circle have some properties. Their common properties like color of the figure, border color, border line, methods to rotate and display the figure are combined together to form a new class *Figure2d*, this class consists of common properties of two-dimensional figures of polygons and circles. So, lower-level hierarchy classes are combined to form a higher-level class.

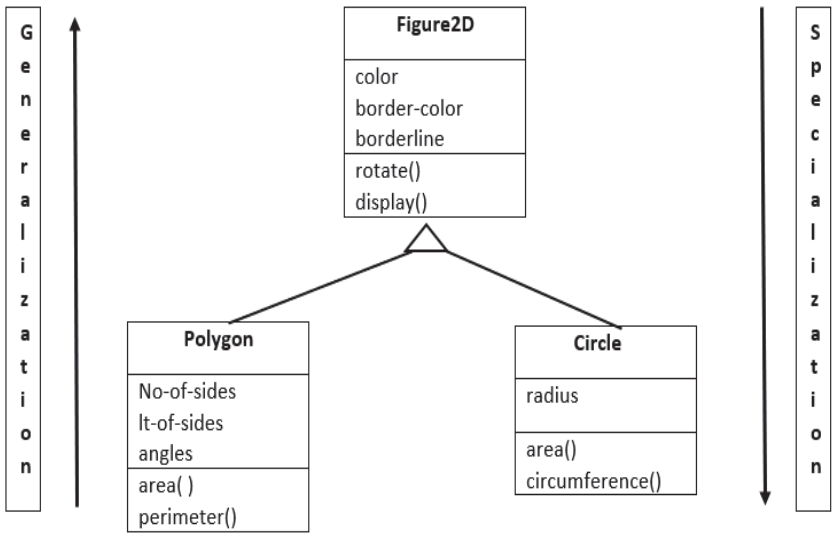


Figure 1.3: Example of generalization and specialization using 3 class hierarchy

If we follow the concept of specialization, the base class **Figure2D** is subdivided into two child classes viz**. Polygon and Circle**. Here, the derived classes can access the properties of the base class. And each derived class contain their distinguished specialized properties.

Space for Learners:

In the object-oriented design phase, we may need to perform generalization and specialization based on the context and demand of the system.

1.11 LINK AND ASSOCIATION

In the next chapter, we will study UML and how we can draw a class diagram, state diagram using UML etc. in OO Design. For this, we need to understand two important parts of object-oriented design to build the relationship among various objects and classes: links and associations.

A **link** is used to connect two or more objects. It is used to establish the relationship between the objects. The link between two objects can be physical or conceptual. Through a link, communication can be performed between two objects. A link is said to be as an instance of an association like we say, an object is an instance of class.

An **association** is used to build the relationship between classes. It is more descriptive in nature. That means, an association can be said a collection of group links sharing a common structure and common semantics. The links in an association are used to connect all the objects in a class. The syntax of link and association are shown here:

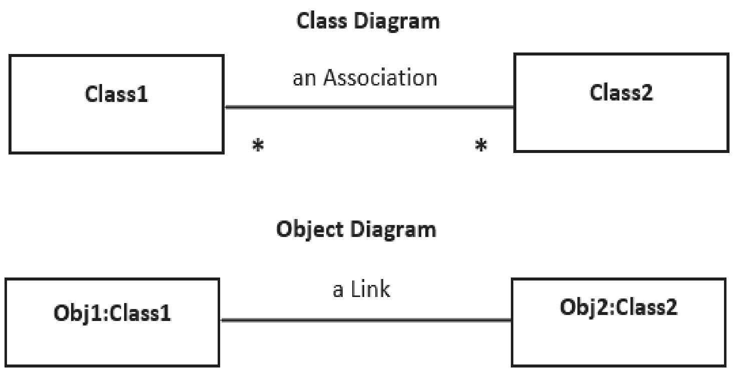


Figure 1.4: Association and Link using Class diagram and object diagram

Let us try to understand the use of links and associations with an example. Consider a model of a teacher, teaching subjects. Suppose, there is two class, Teacher and Subject. And there is an association between the classes as Teacher *teaches* Subject. There is a possibility that a teacher can teach more than one subject, or one can be taught by one or more teachers. So, objects are connected via links. The class diagram and object diagram to show the link and association are presented in following **Figure 1.5**:

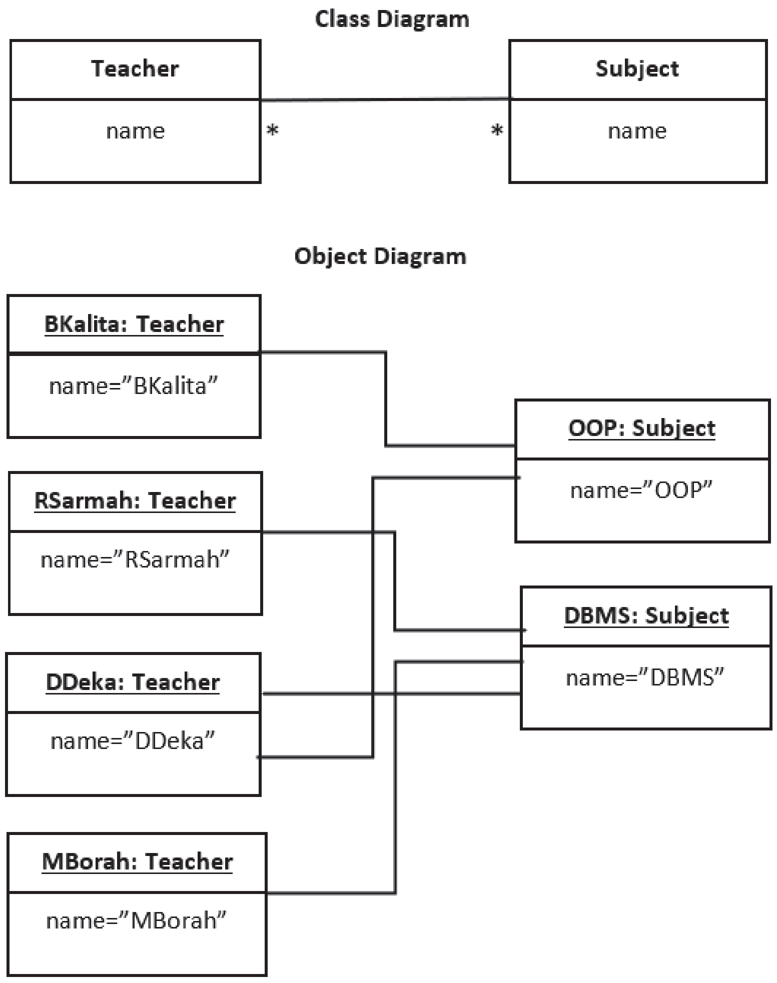


Figure 1.5: Class diagram and object diagram of Teacher and Subject class with many to many relationships to understand links and associations (many-to-many)

1.11.1 Degree of Association

The number of classes that are forming an association is known as degree of association. A degree of association can be of 1, 2, 3 or more.

__Space for Learners:__

Space for Learners:

Unary relationship: Here, the connection is established among the objects of same class.

Binary relationship: Here, the connection is established between objects of two classes. In this course, we will focus on only binary relationships.

N-ary relationship: Here, the connection is established among objects of three or more than three classes.

1.11.2 Multiplicity

M. R. Blaha and J. R. Rumbaugh defined multiplicity in their book [2] as, "multiplicity specifies the number of instances of a class that may relate to a single instance of an associated class. Multiplicity constraints on the number of related objects". In a binary relationship, based on the concept of multiplicity, a relationship can be categorized in mainly three ways:

One-to-one association: Here, a single object of one class builds a relationship with a single object of another class.

One-to-many association: Here, a single object of one class may build relationship with more than one object of other class.

Many-to-many association: Here, multiple objects of the first class may associate with any object of the second class. Again, multiple of objects of the second class may build relationship with any object of the first class. As given example in **Figure 1.5**, a teacher may teach many subjects and a subject may be taught by many teachers. So, it is an example of many-to-many association. Here, the symbol '*' indicates there can be either zero or one or multiple associations can be held between the classes.

-

1.12 SUMMING UP

Space for Learners:

- Object-oriented approach focuses on finding a solution of a problem based on real-life examples.
- Object-oriented approach presents the problem domain in terms of smaller modules known as objects. Objects are consisting of data and procedures.
- Object-oriented software development methodology is composed of three main steps viz. requirement analysis, object-oriented design and implementation of the system.
- The main functions of object-oriented analysis are redefining the problem technically, collection of user and software requirements, identification of the classes and objects, assigning the services to the objects and identification of relations among the objects.
- Object-oriented design plans to build the solution domain based on the output of the object-oriented analysis phase.
- The functions of object-oriented design are identifying the classes based on previous steps, adding new classes on need basis, establishing relationships among them, identification of different constraints and designing the user interfaces.
- Decomposition is a process to divide the whole problem of high complexity into smaller subsystems of lower complexities.
- Object-oriented design can be performed using different models like class model, dynamic model and functional model. In present days, designers are using interaction models instead of functional models.
- Object-oriented programming is to find the solution to the problem based on object-oriented design models.
- Data hiding is a property used in object-oriented programming to hide the internal data (data members, member functions) from unauthorised access.
- Objects are the basic entity of OOP which can be of either physical or conceptual existence.
- A class is a skeleton-like structure combing various data members and member functions. These can be accessed by the instances of the class i.e., objects.
- Abstraction is the process of hiding background detail from the users. Whereas encapsulation makes the system easier

for the users to handle by wrapping up the data and code into a single entity.

- Inheritance is a method through which one class can inherit the properties of another class. The class from which properties are inherited is known as base class and the class which inherits the properties is known as derived class.
- In OOD, relationships between multiple classes are established using associations where multiple objects are established using links. Links are the instances of association.

Space for Learners:

1.13 ANSWER TO CHECK YOUR PROGRESS

1. Class 2.Object 3.Data hiding 4.Object-oriented

5. True 6.False 7. False 8. True

1.14 POSSIBLE QUESTIONS

1. Explain the brief history of evaluation of object-oriented programming.
2. Describe the following concepts of OOP with example: Abstraction, Encapsulation, Polymorphism
3. Define the term link and association. Explain their working in object-oriented design with an example.
4. What are the different types of models mainly used in object-oriented design?
5. Explain the working of object-oriented analysis with suitable diagram.
6. Explain the concepts of generalization and specialization in OOP.
7. What do you mean by degree of an association? What are the different types of associations between the classes in a relationship?
8. What do you mean by one-to-many and many-to-many association? Describe many-to-many association with an example with the help of class diagram and object diagram.

1.15 REFERENCES AND SUGGESTED READINGS

Space for Learners:

1. Booch, G.: Object-Oriented Analysis & Design with Applications. Pearson Education, 1994.
2. Blaha, M.R. and Rumbaugh, J.R.: Object-Oriented Modeling and Design with UML. Pearson, 2007.
3. Balagurusamy, E.: Object-Oriented Programming with C++. Tata McGraw-Hill, 2008 (fourth print).

UNIT 2: OBJECT MODELING TECHNIQUES(OMT) TOOLS

Unit Structure:

2.1 INTRODUCTION

In the previous unit, we have learned the purpose of object-oriented design, different object-oriented approaches and the important concepts related to object-oriented design. With this, we have got the basic idea of different diagrams based on different models of object-oriented design.

In this unit, we will learn how to draw various diagrams using different tools in object-oriented design. In object-oriented modeling, one of the most used methods to perform design is Unified Modeling Language (UML). Different software is available to draw diagrams using UML tools. Object-oriented design is composed of different models. Designers used the models as per the requirement of the problem domain. Each model is represented by specific diagram. Like class models are represented using a class diagram and object diagram. Here, we will learn these design procedures in detail with examples.

2.2 UNIT OBJECTIVES

The main objectives of this unit are:

- To understand the concept of object-oriented modeling
- To know the characteristics and use of UML
- To know various UML tools
- To have a clear concept of different models in object-oriented design
- To be able to draw the class diagram, object diagram, state diagram, DFD as per the problem.
- To have the knowledge of usecase diagrams

2.3 OBJECT-ORIENTED MODELING

Object-Oriented Modeling (OOM) is a process of object-oriented design to design models for developing an application using different object-oriented concepts. A model can be said as an abstraction of a process for understanding its importance before implementing it. The models present how exactly the coding will be done. The execution of a program can be performed using any object-oriented programming language, based on the processes represented in a model.

The detailed design in object-oriented modeling consists of several phases. In the first phase, the models look more abstract since it focuses on the external detail of the system. Then, the model evolves at different phases, and more details are included like how the internal processes will be implemented, what functions are required and how the entire system will be built.

The main reason for designing a model using an object-oriented approach before starting the actual implementation or coding is that:

Easy to Understand: The diagrammatic representation of any system is easy to understand for the users. Understanding the complete coding procedure is not easy for normal users. Model diagrams help user like clients, end-user, stakeholders to give feedback to the developer team as how they want the exact system should work.

Abstraction: To have a clear idea of the system requirements and the input-output of the system.

2.4 UML

Unified Modeling Language (UML) is a widely used modeling language to design any software. It is used to identify, picturise, create and document the models of any software system. UML was initially created to use notational systems in software design. It is adopted as a standard software design tool by the Object Management Group (OMG). Further, UML was also approved as an ISO standard by International Organization for Standardization (ISO). With time, different versions of UML were released and it is widely used by software designers over the world.

UML is widely used to model software systems. The main artifact in designing a model using Unified Modeling Language is an object. Using UML tools, different models like class model, object model, state model can be designed. These models can be either static or dynamic. The object-oriented design is performed using UML. So, understanding the concepts of object-oriented design is a prerequisite to use UML. As per the requirements, the object-oriented design approach is transformed into any form of UML diagram.

2.4.1 Goals/ Characteristics of UML

The main characteristics of UML are:

- UML is a General-Purpose modelling language.
- It designs diagrams based on the output of the object-oriented analysis (OOA) approach.
- The workflow of a system to be developed is visualized by UML.
- Both static and dynamic models can be presented using UML tools.
- UML tools are easy to use, understand and helps to design models conveniently.

2.5 DIFFERENT MODELS OF OOD

In this section, we will learn the working of different models of object-oriented design, their respective diagrams in detail. With this, we will study how to draw these diagrams based on the requirement using Unified Modeling Language (UML) tools.

2.6 CLASS MODEL

A class model is a static model which represents the structure of the system in terms of classes. The class model contains the classes, their properties, the association among the classes. The properties of classes contain different attributes and operations. These are also referred with the term data member and member functions. Associations among the classes are performed using the concept of generalization and specialization.

During the study of class models, we need to have clear concepts on the various concepts on class like enumeration, multiplicity, scope and visibility of the classes.

2.6.1 Class Diagram

As the name suggests, in object-oriented design, the class diagrams are used to represent the classes identified in the object-oriented analysis phase.

Class diagram presents different graphic notations for modeling the classes, showing their relationships and describing the objects. It is a static diagram. The advantages of class diagrams are:

- It is easy to understand and concise.
- It worked well for practice and plan easily.
- It has mapped to popular OOP languages like C++, JAVA, etc.
- It helps programmers to implement the program easily based on the model design.
- It is useful for abstract modeling.

The syntax of defining a class on a class diagram is shown in Figure 2.1. It is represented using a rectangular box consisting of three blocks. The class name is written on the top of the box. In the second block, the properties of the class are written known as attributes.

In object-oriented programming, these attributes are mentioned as data members or variables. The data types and other properties of the data members can also be written in the class diagram. The methods known as member functions are defined in the last block of the rectangular box. Let us try to learn how to draw class diagrams for two different scenarios.

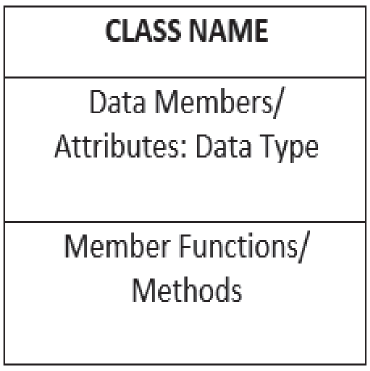

CLASS NAME
Data Members/ Attributes: Data Type
Member Functions/ Methods

Figure 2.1: Syntax of representation of a class

Scenario 1: *In a company, employees are working in departments. A department can have multiple employees, but one employee can work in a single department. One department can control multiple projects. An employee is categorized as either 'Manager; or as 'Other Employee'. Different information as per the requirement can be stored in classes as attributes.*

So, the class diagram using the above scenario can be drawn as shown in Figure 2.2. The diagram consists of three main classes, where the class **Department** can be considered as the base class. The **Employee** class is associated with the **Department** class with '***works on***' relationship. **Department** class stores the name of the department as an attribute of string data type. The **Employee** class stores the employee ID and name of the employee as shown in the figure. Further, the concept of specialization is used in the **Employee** class to categorized it to **Manager** class or **Other_emp** class. **Department** class is associated with the

Project class as department undertakes the different projects. **Project** class stores the value of two data members, *project_name* and *duration*.

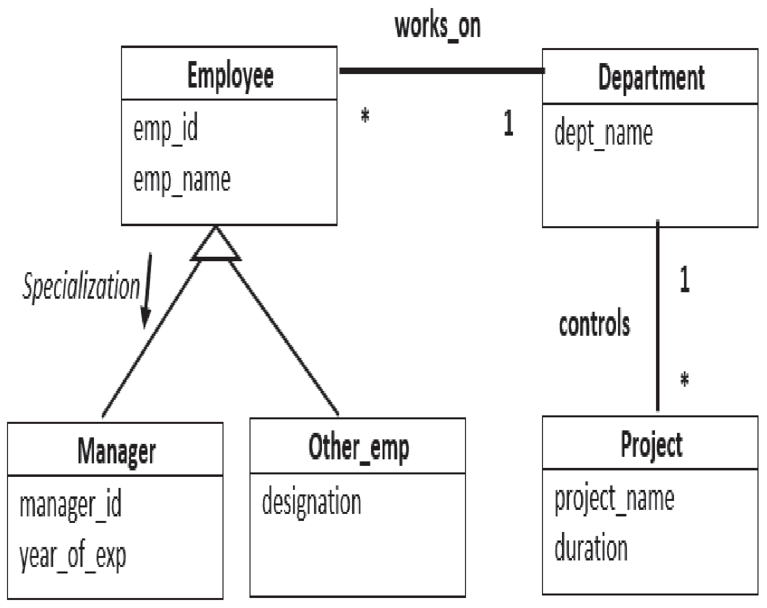


Figure 2.2: Class diagram of a company system as in scenario 1

Scenario 2: *Let us consider a basic system of online shopping. Customers can order multiple times, where order number is issued against each order. Customers can select payment option against each order. The payment system will generate bill number, mode of payment, amount and date of product. Payment class is further subdivided based on the payment options like internet banking, card payment and cash of delivery.*

Figure 2.3 shows the class diagram of basic online shopping system as described in scenario 2. After identification of classes, desirable attributes, their data types and methods of each class are also shown.

Space for Learners:

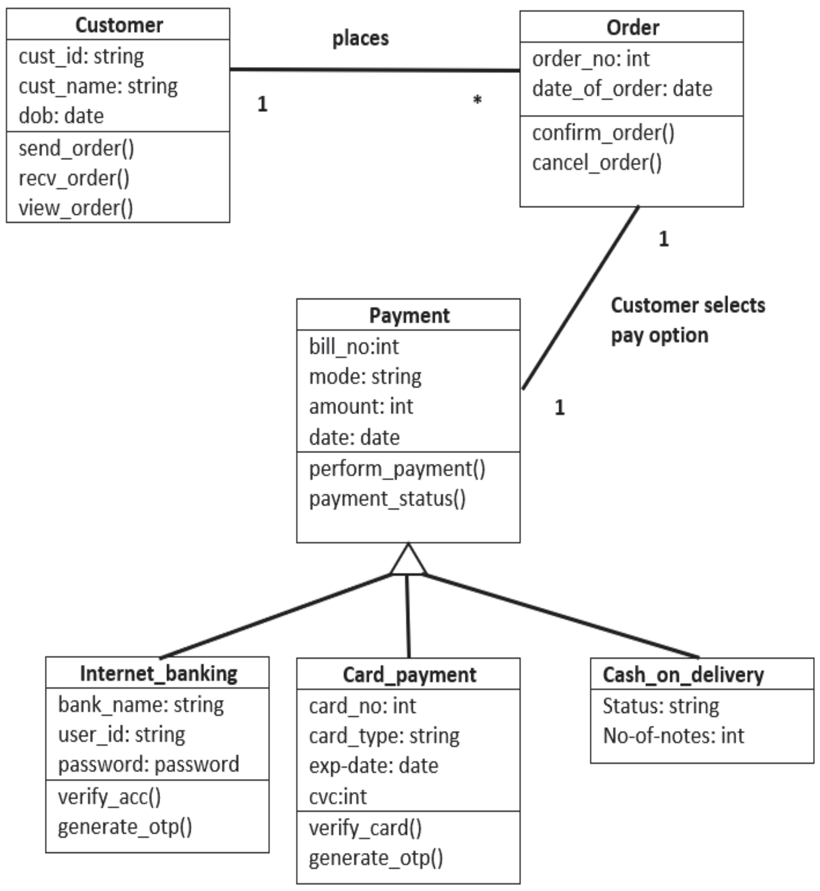


Figure 2.3: Class diagram of a basic online shopping system as in scenario 2

2.6.2 Object Diagram

As objects are the instances of a class, object diagrams are also instances of class diagram. It is derived from the class diagram. So, the object diagram is fully dependent on class diagrams.

Object diagrams are almost similar to the class diagrams. The key difference between the class diagram and the object diagram is that class diagram represents an abstract view of the systems showing relationships between the classes. Whereas, object diagram represents the instances of the classes at any particular point of time.

The syntax of declaring an object in an object diagram is shown in Figure 2.4. The objects in the object diagram are instantiated from respective classes. The object diagram shows the value assigned to the attributes defined in the class. There can be more than one objects of a class.

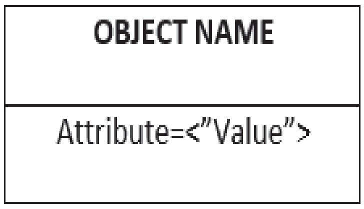


Figure2.4: Syntax of an object in object diagram

The object diagram shows the links among different objects. The objects of the same classes can be linked in the diagram. The links among the objects of different classes are also presented in an object diagram. There can be more than one object diagram for a class diagram.

Figure 2.5 shows the object diagram of a company management system as described in scenario 1. This object diagram is instantiated from the class diagram presented in Figure 2.2.

A simple object diagram of the online shopping management system is shown in Figure 2.6. It is instantiated from Figure 2.3. Here, a single object is declared for each class; These objects are linked based on the associations of classes. The values assigned to the attributes are defined in the class diagram. Like this, we can draw any object diagrams from a class diagram.

Space for Learners:

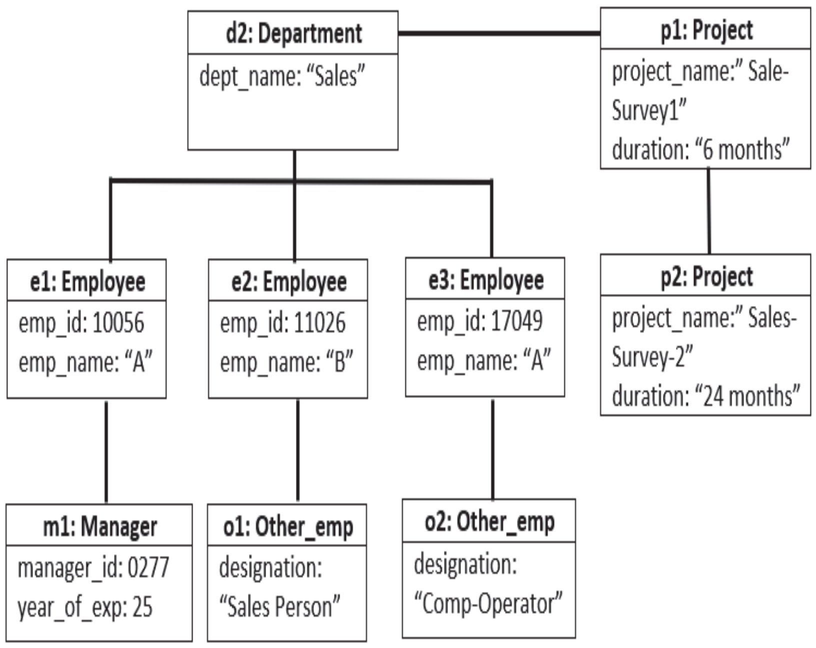


Figure 2.5: Object diagram of Company management system for class diagram shown in Figure 2.2

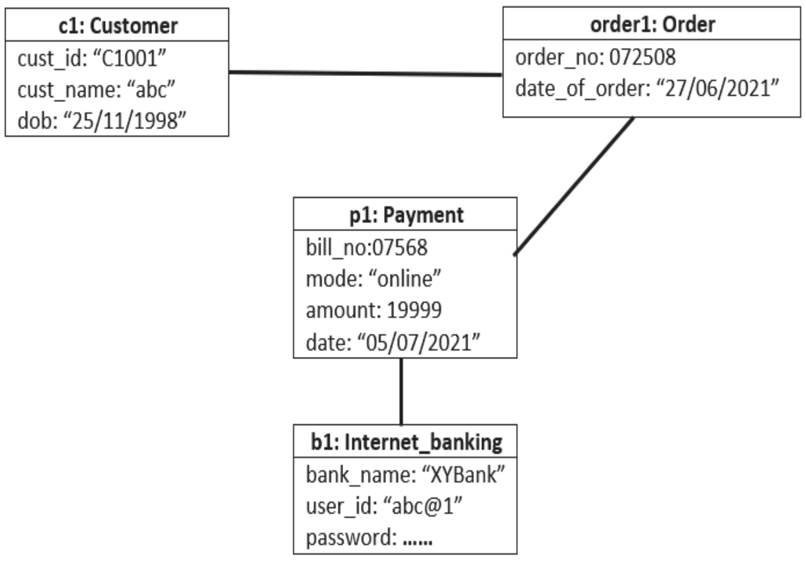


Figure 2.6: Object diagram of Online Shopping System for class diagram shown in Figure 2.3

Space for Learners:

CHECK YOUR PROGRESS - I

Fill in the blanks:

1. _____________ is widely used to model software systems.
2. A __________a static model which represents the structure of the system in terms of classes.
3. Class diagram is a ________________ diagram.
4. Object diagram is fully dependent on _____________

Space for Learners:

2.7 DYNAMIC MODEL OR STATE MODEL

State models are known as dynamic model in OOD. State model represents the dynamic behaviour of a system. A state diagram is a collection of different states of the system which occur following a series of events.

A state can be said as an abstraction of the attribute values and links among the objects. Different values and links are grouped together to form the state model which shows the dynamic behaviour of the system. In UML, an event is an action occurring at a particular point of time. In a state diagram, the states are changed based on the events as per the need of the system. It changes the states during different phases of the system. Both the states and events in a dynamic model depend on the level of abstraction.

A transition is known as the change from one state to other. The change of state can be understood by the example of the physical changes of water. It is normally in liquid state. If we freeze it, for some time it changes to a solid state. If we start melting it, it returns to liquid state again. If the water is boiled for more times, it transforms into gaseous state. Similar things can be happened with a software model, which changes its state as per the requirement.

2.7.1 State Diagram

Dynamic models or state models can be represented using the state diagram. The state diagram presents the transition or change in the state based on the events occurring in a system. The number of states in a state diagram is finite. The main components of state diagrams are:

States: The states are represented using a rectangle symbol with rounded corner.

Initial state: the starting state of state diagram is known as initial state. It is represented using black circle.

Transition: The transition or change in state is shown by using an arrow. The arrow symbol is labelled by the event name.

Self transition: An arrow pointing to the same state is used for self-transition.

Final state: In a state diagram, the conclusive state is represented a filled circle with a circle notation.

The notations using UML of the state diagram are:

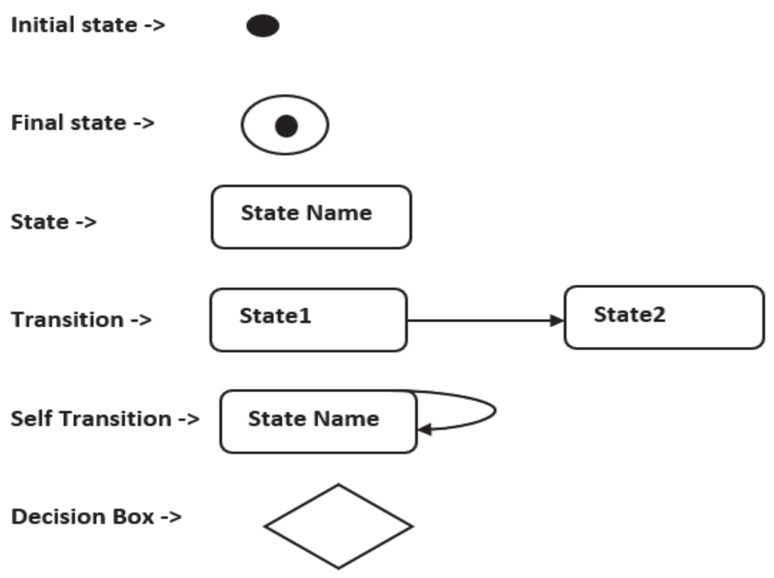


Figure 2.7 shows a partially completed state diagram of the household motor control system as derived in [2]. We try to understand the design of state diagram with this example. A motor primarily in *Off* state, when it is switched on, it changes its state to

Starting. That is, it is preparing for run. Then, it to transforms to *Running* state and continue to run. From this, if we switched off the motor, it will go to *Off* state again. And if it is continued to run, it may go to *Too Hot* state and the overheating may harm the functions of the machine.

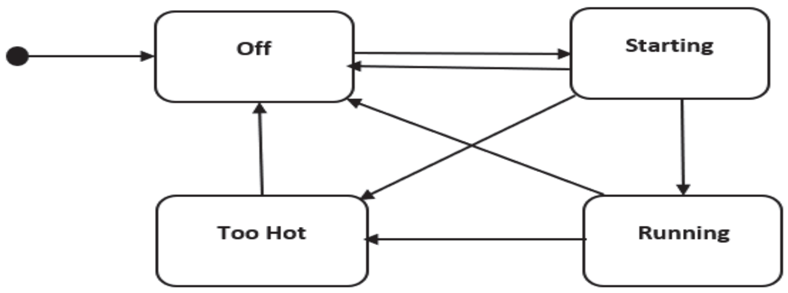
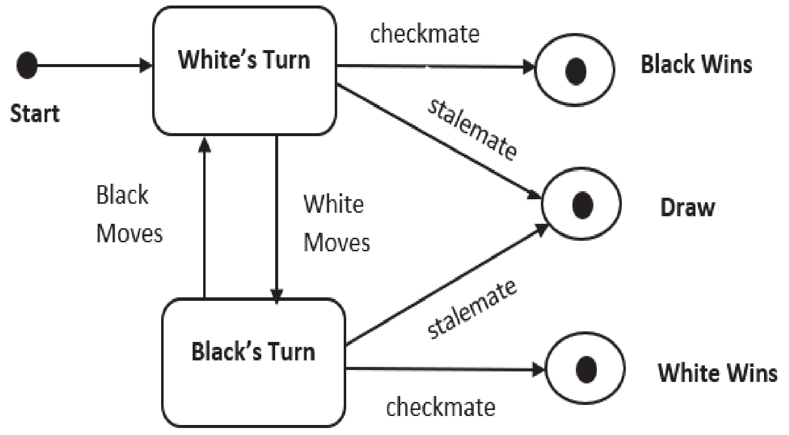


Figure 2.7: Partially completed State diagram of household motor control system



-

Figure 2.8: State diagram of a Chess Game

Let us consider another example as shown in Figure 2.8 which represents the state diagram for a chess game. This example is derived from [2]. This diagram consists of mainly two states namely *White's Turn* and *Black's Turn*. The states are changed continuously in every move which starts from the *White's Turn* first. Depending on the checkmate and stalemate, the results of the match is decided, So, the final state of the game would be

Space for Learners:

either *Black Wins* or *White Wins* or *Draw*. Thus, we can draw the state diagrams of a system.

2.8 FUNCTIONAL MODEL

In object-oriented analysis and design, the functional model specifies the overview of the entire system. It provides the overall summary of the functions that the system is going to perform. This model presents the system based on main three tasks: inputs of the system, processing and outputs of the system. To represent the functional model of a system, Data Flow diagram (DFD) is used. DFD helps to provide functions of the internal process of the system.

2.8.1 Data Flow Diagram (DFD)

Data Flow Diagram (DFD) represents the functions of the system in form of the inputs to be provided to the system, the processing parts of the system based on the inputs and the desired output of the system along with the detail of internal data stores in the system. DFD can be drawn in many phases or levels. These levels arenamedasLevel-0 (Context Diagram), Level-1, etc. The basic components of DFD are:

External Entity: These entities are named based on real-life objects. These entities take the information and submit to the processes. It can ask any query to the process and gets information or response from the process.

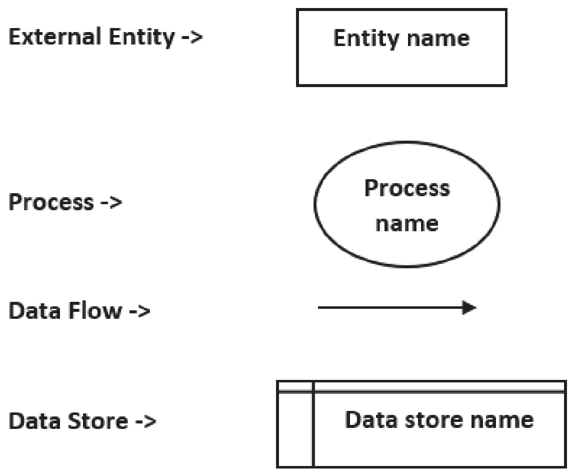
Processes: In DFD, a process is the activity that executes the data flow of a system. DFD shows how the data flows through a process in a system. A process can have multiple sub-process under it.

Data Flows: Data flow depicts the movement of information among external entities, processes and the place where data is stored through arrows.

Data Store: Data store represents the place where data is stored. It does not perform any operations, simply stores the data. Processes can insert, update, delete or retrieve data to Data store.

Space for Learners:

The notations used in DFD are:



Let us consider an example of DFD of basic *Store Management System* as shown in Figure 2.9. This system consists of only two entities, *Customer* and *Seller*. Here, the seller or shopkeeper would enter product details which would be stored in the database. *Customer* has to register into the system. After successful registration, a customer gets Customer ID. A customer can order products, if the product is available, a customer will get order confirmation message. This Process is further subdivided into two sub processes namely *Registration* and *Selling Process* as shown in Figure 2.10.

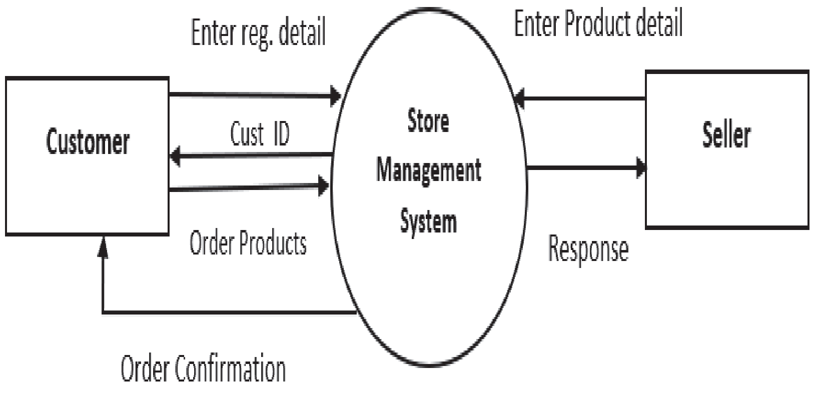


Figure 2.9: Level-0 DFD for basic store management system

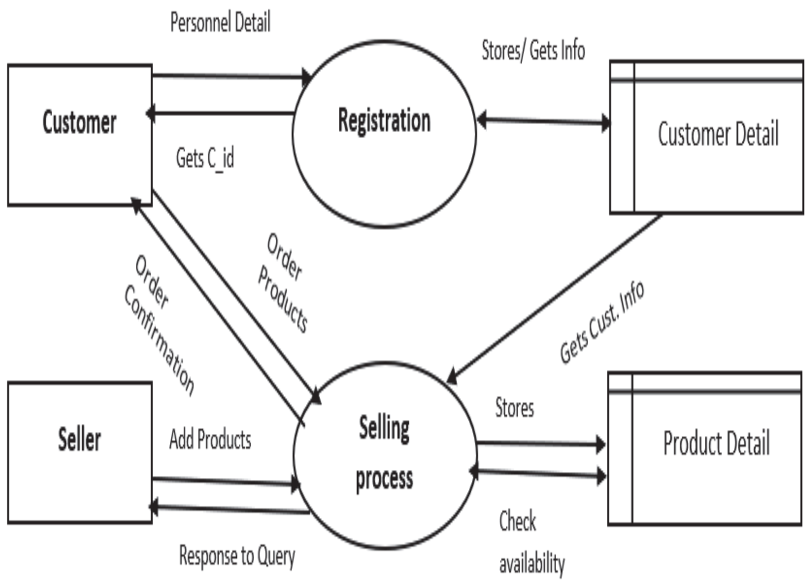


Figure 2.10: Level-1 DFD for basic store management system

CHECK YOUR PROGRESS - II

State True or False:

5. A transition is known as the change from one state to other.
6. The states and events in a dynamic model do not depend on the level of abstraction.
7. Dynamic models cannot be represented using the state diagram.
8. DFD is used to represent the functional model of a system.

2.9 INTERACTION MODEL

The Interaction model describes the interactions inside a system. The class model presents the objects present in the system; the state model presents the life-cycle of declared objects in terms of states whereas as interaction model presents how the interaction takes place among these objects. Interaction models can be represented

Space for Learners:

using different diagrams like use case diagram, sequence diagram etc. Here, we will study the designing of use case diagram with an example.

Space for Learners:

2.9.1 Use Case Diagram

A use case is a representation of how a human being (known as actor) uses a process to reach his goal. Use case is a part of functionality that a system is able to provide by interacting with the actors. An actor is the person who act as a direct external user in a system. A use case is composed of one or more actors. Use case is a description of the role of each actor to accomplish particular goal using various processes. The use case of a system can be represented by a use case diagram.

Let us take an example of the operation of a vending machine as presented in [2]. The use case of the scenario is buying a beverage by a customer. The customer delivers the beverage after successful selection and payment made by customer. Figure 2.11 shows the use case diagram for this vending machine. It represents three actors namely Customer, Repair Technician and Store Clerk. Inside the rectangular box, four use cases are shown in which actors participate. The use cases are *Buy beverage, Performed Maintenance, Make repairs* and *Load Items*. Repair Technician participates in two use cases whereas other actors participate in one use case.

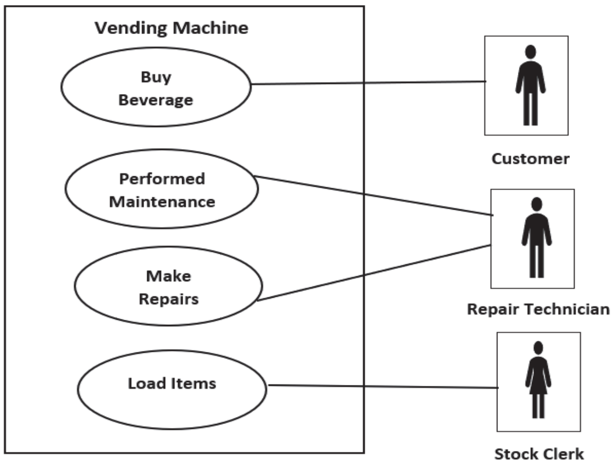


Figure 2.11: Use case diagram for a vending machine

In this unit, we have learnt different object-oriented modeling techniques to design a system. Each model type is represented by a particular diagram. These diagrams help us easy to understand the workflow and behaviour of the system. Programmers perform the coding to execute the system by following these designed models. So, this is an important phase in developing a system using object-oriented concepts.

2.10 SUMMING UP

- Object-oriented modeling (OOM) is a process of designing a model to develop a system using different object-oriented concepts.
- A model is an abstraction of a process to understand it before its implementation. The system is built following the model.
- Different models are there to design a system like class model, state or dynamic model, functional model etc.
- UML is mostly used general-purpose modeling language to identify, picturise, creation and documentation the models of any software system.
- The Class model is a static model to represent the structure of the whole system in terms of classes. The classes contain their different characteristics and methods.
- A Class model is represented using class and object diagrams.
- A Class diagram shows the classes, their relationships, the attributes and methods present in each class.
- An object diagram is an instantiation of the class diagram. A class can have more objects. Object diagram shows the links among different objects.
- Each object in an object diagram contains particular value of all attributes or data members of its class.
- The dynamic model or state model shows the dynamic behaviour of a system in terms of its change of state following a series of events.
- A state is nothing but an abstraction of attributes and links among the objects. Transition is change of a state to another in a system.

Space for Learners:

- A state model is represented by a state diagram. The main components of state diagram are initial state, final state, other states and transitions.
- Functional model specifies the overview of the whole system inputs, its processing and outputs to the system.
- Functional models are represented using Data Flow Diagram (DFD) in a system.

2.11 ANSWERS TO CHECK YOUR PROGRESS

1.UML 2. Class model3.Static4. Class diagram

5. True 6. False 7.False 8.True

2.12 POSSIBLE QUESTIONS

1. Explain the concepts and importance of object-oriented modeling.
2. Describe the following models of OOD with example: Class model, Dynamic model, Functional model
3. Explain the important characteristics of UML.
4. Explain the working of class diagram and object diagram with a suitable example.
5. Why the dynamic models are required in object-oriented modeling? How it is different from a class model.
6. How can we draw a state diagram of a system? Explain with a suitable example.
7. What are the basic components of an DFD? Draw a data flow diagram of level-0 DFD for online shopping management system.
8. Prepare a class diagram of your family tree with name and age as attribute of each class. Draw an object diagram of this class diagram.
9. Draw a state diagram of complete telephone call procedure with its different states.

10. Explain the working of an interaction model.

11. Write the importance of use case diagram. Draw a use case diagram of simple vending machine with explanation of its working.

Space for Learners:

2.13 REFERENCES AND SUGGESTED READINGS

1. Booch, G.: Object-Oriented Analysis & Design with Applications. Pearson Education, 1994.
2. Blaha, M.R. and Rumbaugh, J.R.: Object-Oriented Modeling and Design with UML. Pearson, 2007.
3. Balagurusamy, E.: Object-Oriented Programming with C++. Tata McGraw-Hill, 2008 (fourth print).

UNIT 3: PHASES OF OBJECT-ORIENTED DEVELOPMENT

Space for Learners:

Unit Structure:

3.1 INTRODUCTION

In software development, most of the methods used are based on functional or data-driven approach. Object-oriented approach is different from these approaches in many ways. In object-oriented methods, data and functions are integrated in to one group. It develops the software from some self-contained modules known as objects. These objects can be easily modified, replaced and reused. Object-oriented approach works based on the concepts of real-world systems for which it becomes a popular methodology for programmers. In object-oriented methodology, software can be treated as a collection of discrete objects in which data and the operations grouped together for desired outputs.

3.2 UNIT OBJECTIVES

The main objective of this chapter is to

- introduce the concept of object-oriented modelling.
- give an overview of object-oriented methodologies.
- introduces object-oriented methodology and
- discuss the phases of object-oriented design in detail.

3.3 OBJECT ORIENTED MODELLING

A model in object-oriented approach can be said as abstract view of the problem with an objective to understand its purpose before implementation. Since, working of a real system is very complex, so we need to simplify it. A model hides the exact working procedure and shows the important characteristics of the problem.

Through a model the problem is conceptualized and presented distinctly. We can say that modelling helps us to deal with complexities of the system and makes the whole system easily understandable.

In object-oriented development, a model is an iterative process. In designing the model as the working of the system progresses, more detail is added to the model. Models are designed in different levels. In every level, a model can be subdivided into some smaller models for better understanding of their purposes.

3.4 OBJECT ORIENTED METHODOLOGIES

Object-oriented methodology is completely based on the concepts of classes and objects. These concepts are introduced based on the real-life examples. A class can be skeleton of a working procedure. An object is an instance of a class. We have discussed these in previous units.

An object can be created, modified, categorized, merged, described, removed or reused. In object-oriented development, software components can be modified and reused easily. It results in high quality, higher productivity and lower maintenance cost of the software.

Object-oriented development uses different object-oriented techniques to analyse, design, implement and maintenance of the system. The developer determines what are the objects, their characteristics, responsibilities and relationships with other objects.

In object-oriented design phase, the whole architecture of the system is described. The objects of different class and their relationships are shown. In implementation phase, actual coding is performed by the programmer based on the design report. The program is connected with a database to store data so that the whole system becomes operational.

3.5 OBJECT ORIENTED PROCESS

As we studied earlier, object-oriented development is built with its basic element as object. For this, first the detail analysis is performed to collect all the requirements. Object-oriented development can be said as the traditional approach of system designing as it also follows a sequential process to design the system. The fundamental steps for designing a system using object-oriented process are:

- System Analysis
- System Design
- Object Design
- Implementation

3.5.1 System Analysis

Like any other design system, the first phase of object-oriented development is system analysis. Here, the interactions between the developer and the clients or the user take place. Developer collects the requirements and analyse them to have a clear idea of the system.

Based on the information gathered in the analysis phase, a document is prepared in which detail requirement of the system is specified. This documentation describes the different models and their functioning. In this stage, the implementation details are not examined. We will discuss it in detail in section 3.6.

Space for Learners:

3.5.2 System Design

The second step of the object-oriented process is system design. In this step, the entire system is designed based on requirement analysis and specification documents of the previous step. In this phase the system is divided into smaller parts known as sub-systems. The sub-systems interact among themselves to solve the problems. We will discuss it in detail in section3.7.

3.5.3 Object Design

In this phase, the details of the system analysis and system design are implemented. The Objects identified in the system design phase are designed in this phase. Here the implementation of these objects is decided in the form of data structures and the relationships between the objects are designed. For example, we can define a class student and then we can create several objects of this type.

In this phase, of the designer decides about the classes in the system based on the system requirements. The designer also decides, whether the classes need to be created freshly or inherited from existing classes. We will discuss it in detail in section 3.8.

3.5.4 Implementation

During this phase, the class objects and the interrelationships of these classes are translated into actual code by using an object-oriented programming language. The required databases are created and the complete system is transformed into operational one.

3.6 SYSTEM ANALYSIS

In the system analysis or object-oriented analysis phase of software development, the system requirements are determined, the classes and the relationships among classes are identified.

There are three different analysis techniques used in conjunction with each other in object-oriented analysis. They are object modelling, dynamic modelling and functional modelling.

3.6.1 Object Model

The object model describes the structure of the objects. It describes the identity of the objects, relationships with the other objects, attributes, and the operations. The object model shows the primary view about how real-world problems are divided into objects and the objects interacts with each other. The object model provides the basic framework on which other models are positioned.

The object model is represented graphically by an object diagram. The object diagram contains the classes interconnected by lines. Each class represents a set of individual objects. The association lines establish relationships among classes. Each association line represents a set of links from the object of one class to the object of another class.

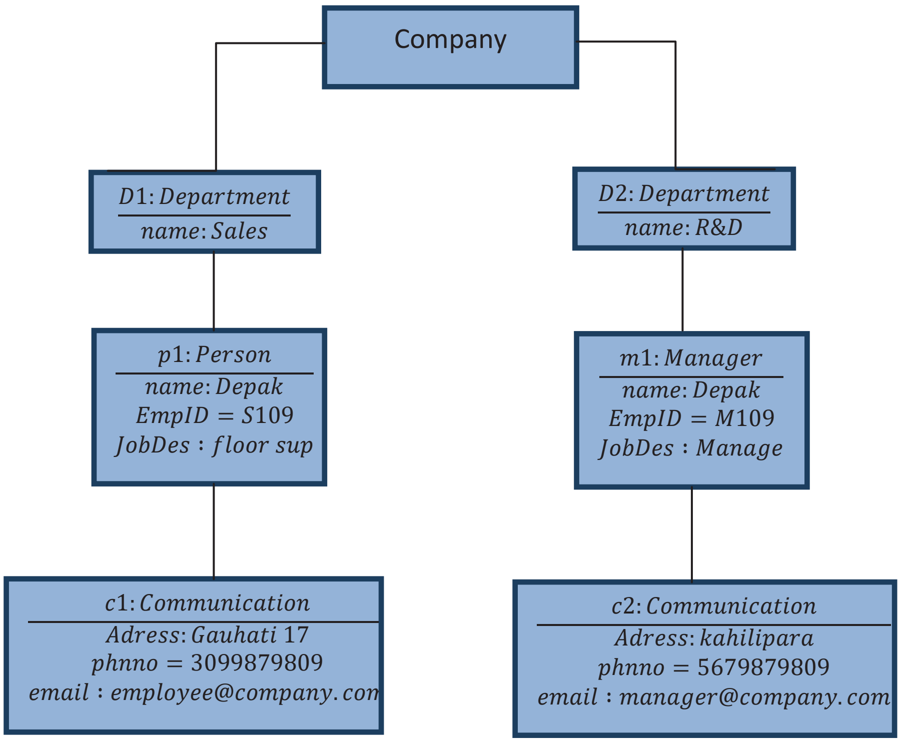


Fig3.1: Object Diagram

Space for Learners:

Space for Learners:

A class describes a collection of similar objects. It is a template where certain basic characteristics of a set of objects are defined. A class defines the basic attributes and the operations of the objects of that type. For creation of objects, instances of the class are to be created as per the requirement of the system.

Classes are built on the basis of abstraction, where a set of similar objects with common characteristics are listed. The characteristics concern to the system under observation are taken and the class definition is made.

3.6.2 Dynamic Modelling

Dynamic model describes about the features that changes with the time. It is used to specify and implement control features of the system. It describes states, transitions, events and actions. The dynamic model includes event trace diagrams to describe a scenario. An event is a task from one object to another, which occurs at a particular time. An event is a one-way transmission of information from one object to another. A scenario is a sequence of events that occurs during one particular execution. Each of the basic execution of the system is represented as a scenario. In the fig 3.2, a scenario of ATM cash withdrawal is explained.

The dynamic model is represented graphically by state diagrams. A state corresponds to the interval between two events received by an object and describes the "value" of the object for that time period. A state is an abstraction of an object's attribute values and links, where sets of values are grouped together into a state according to properties that affect the general behaviour of the object. Each state diagram shows the state and event sequences permitted in a system for one object class. The state diagram should follow the object-oriented development notation. The outcomes of dynamic modelling are event-trace diagram and state diagram.

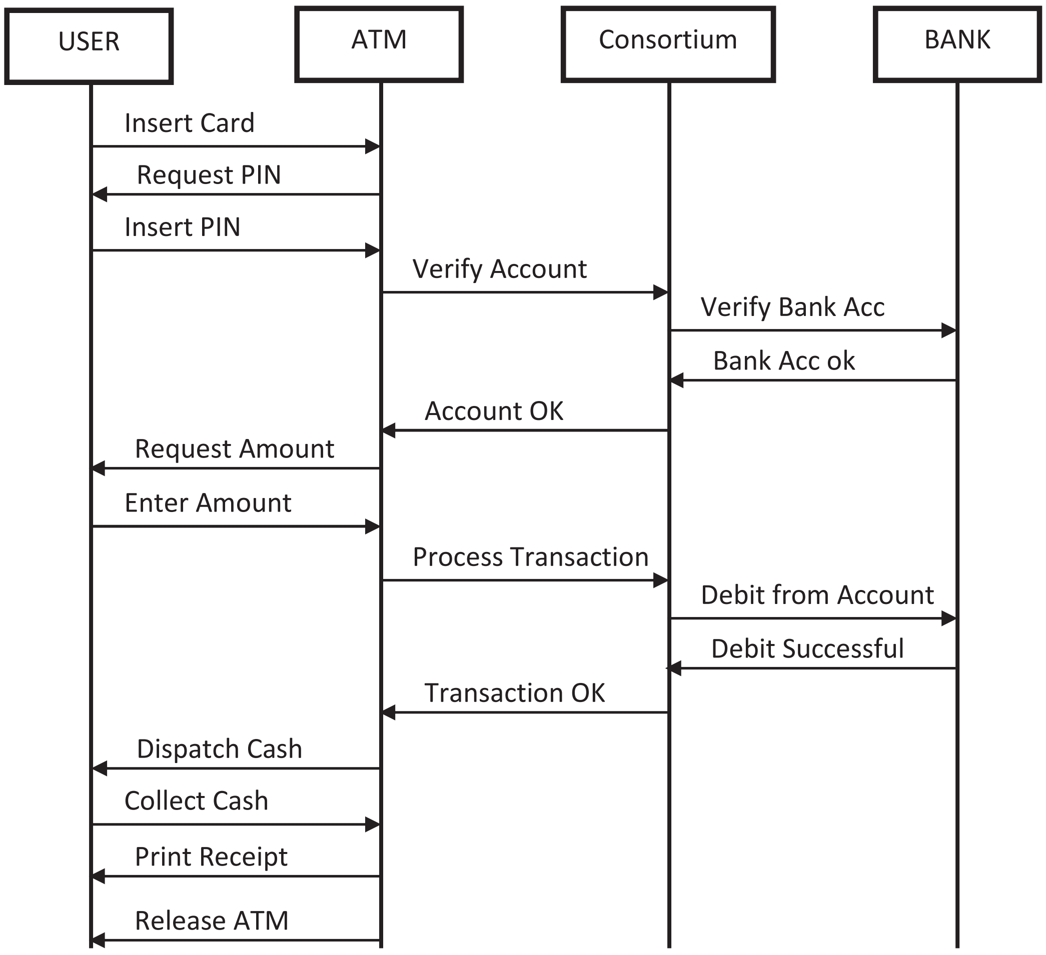


Fig3.2: Event Trace Diagram

3.6.3 Functional Modelling

The functional model describes computations and specifies those aspects of the system concerned with transformations of values. Functions, mappings, constraints, and functional dependencies are described in this model. The functional model describes what the system does, without any detail about how and when it is done.

The functional model is represented graphically with data flow diagrams, which show the flow of values from external inputs, through operations and internal data storage, to external outputs. Data flow diagrams show the dependencies between values and computation of results from the input with the help of functions. Functions are invoked as actions in the dynamic model and are shown as operations on objects in the object model. Data flow diagrams are discussed in detail in section 2.8.1.

3.7 SYSTEM DESIGN

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system. It is a process where system developers design the overall structure of the system. System design emphasises on how to solve the problem. The system design is divided into two parts – logical design and physical design.

The logical design of the system is an abstract representation of the inputs, outputs and data flows of the system. This is often conducted by modelling; modelling involves a diagrammatical representation of an actual system. System design includes Data flow diagram, Class Diagram, Entity-relationship diagrams etc.

The physical design relates to the actual input and output process of the system. Here how data is provided to the system, how it is authenticated, how it is processed, and how it is displayed are defined.

In object-oriented design methodology, system design is one of the phases of the software development life cycle. During this phase, developers decide the overall structure and style of the system. During system design, the following design decisions are to be handled.

a) Estimation of system performance
b) Make a reuse plan.
c) Organize the system into subsystems
d) Identify concurrency inherent in the problem
e) Allocate subsystems to hardware.
f) Estimating Hardware Resource Requirements
g) Manage data stores.
h) Handle global resources.
i) Choose a global control strategy
j) Handle boundary conditions
k) Set trade off priorities
l) Select an architectural style.

a) Estimating System Performance

A rough performance estimates is calculated by the system. The purpose of this step is to check feasibility of the system. The execution of the system should be fast and free from common errors. In this phase, number of transactions to be processed by the system, response time needed, storage requirements etc are estimated.

b) Making a Reuse Plan

Reuse is one of the main advantages of object-oriented methodology. There are two different types of reuses, using existing things or creating new reusable things. It is easier to reuse existing things than to design new things. Reusable things can be models, libraries, frameworks and patterns. The logic in a model can applied to multiple problems.

c) Organizing a system into Subsystems

A subsystem can be defined as a group of classes, associations, operations, events and constraints that are interrelated and can be used to solve a large problem. For example, file system is a subsystem of operating system. A system may be divided into smaller subsystems and each subsystem may further be divided in to smaller subsystems without affecting to the output result.

d) Identifying Concurrency

Concurrency is an important issue that needs to be addressed in design process as it may affect the design of classes and their interfaces. Concurrency is very important for improving the efficiency of the system. One important goal of the system design is to identify the objects that must work concurrently and synchronize the objects that have mutual activity.

e) Allocation of Subsystems

We must allocate hardware to the each of the subsystems, that is either a general-purpose processor or specialized functional unit. The system designers must ensure the following:

- Estimate performance and allocate the resources needed by system.

Space for Learners:

- Choose hardware and software to implement the subsystems.
- Allocate software processors such that it satisfies performance and minimize inter process communication
- Determine the connectivity of the physical units related to the subsystems.
- Define the connection between nodes and communication protocols to be used.
- Consider the needs for redundant processing and provide infrastructures.
- Identify the interfaces applied in deployment.

UML deployment diagram can be used to present the above steps. A deployment diagram shows how the systems will be physically distributed on the hardware.

f) Estimating Hardware Resource Requirements

To increase the performance of system, multiple processors or hardware may be used. The number of processors required depends on the computation requirement and the speed of the machine. The system designer estimates the requirement of processing power for a steady computing and high performance.

While implementing the system a decision needs to be made regarding which subsystems will be implemented on which set of hardware and software. In implementing the subsystems in hardware following are needs to be keeps in mind

- Cost: As the technology advances, the cost of hardware has come down still in system design the designer should keep an eye on the cost of implementation and hardware requirement.
- Performance: In this phase system designer need to ensure high performance system.

g) Management of Data Storage

System designer needs to decide on the data storage for implementation of the data structures, files and databases used in the system. In identifying the data storage complexity of the data, the size of the data, the access method, access time and portability needs to be kept in mind. Having considered these issues, the designer must take a decision about whether data can be stored in flat files or in relational databases or in object databases.

h) Handling Global Resources

The system designer must identify the global resources and determine a mechanism for control. There are several kinds of global resources:

- Physical system: processors, tape drives and communication channels.
- Input, output: keyboard, mouse, display screen etc.
- Logical Ids: object IDs, filenames, and class names.
- Access to shared data: Databases

i) Choosing a Software Control Strategy

In designing the system, it is best to choose a single control style for the entire system. There are two kinds of control flows in a software system: External control and Internal control. External control concerns the flow of externally visible events among the objects in the system and the internal control refers to the flow of control within a process. It exists only in the implementation and therefore is neither inherently concurrent nor sequential.

There are three kinds of external events: procedural-driven sequential, event-driven sequential and concurrent. In a procedure-driven system, the control lies within the program code. Procedures request external input and then wait for it, when input arrives, control resumes within the procedure that made the call. In event-driven, the developers attach application procedures to events and the dispatcher calls the procedures when the corresponding events occur. In concurrent control, the developers need to ensure concurrent execution of the system.

j) Handling boundary Conditions

In designing of the system, the system designer must consider boundary conditions also need to address the issues like initialization, termination and failure.

- Initialization: It refers to initialization of constant data, parameters, global variables, tasks, guardian objects and classes as per their hierarchy.
- Termination: Termination is required for all the subsystems to release the reserved resources. In case of

Space for Learners:

concurrent system, a task must intimate other tasks about its termination.

- Failure: It is the unplanned termination of the system, which can occur due to various reasons such as system fault, wrong user input, due to exhaustion of system resources, external breakdown or bugs in system. A good design must not affect remaining environment in case of any failure and there must be a mechanism for recording details of system activities and error logs.

Space for Learners:

k) Setting trade-off Priorities

The system designer must set priorities for each of the subsystems that must be used as a guide trade-off for the rest of the design. For example, system can be made faster by adding extra memory. Design trade-offs involve not only the hardware but also the process of developing the system. System designer must define the importance of the various criteria as a guide to design trade-offs. Design trade-offs affect entire character of the system. Priorities are generally specified as a statement of design philosophy.

l) Architectural Styles

Software architecture, according to ANSI/IEEE Standard 1471-2000, is defined as the "fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution."

The architecture of a system describes its gross structure. This structure illuminates the top-level design decisions, including things such as how the system is composed of interacting parts, where are the main pathways of interaction, and what are the key properties of the parts. Additionally, an architectural description includes sufficient information to allow high-level analysis and critical appraisal.

Space for Learners:

3.8 OBJECT DESIGN

Object design is the third phase in the Object-Oriented Development. In object design, the designer adds more details and refinement to the system. In object design, the designer implements the objects discovered in analysis phase.

The operations identified in analysis phase are expressed as algorithms and internal operations. The classes, attributes and associations found in analysis phase are implemented with specific data structures. If required, new object, classes can be added in this phase. The following steps are performed in the object design phase:

- Combine all the three models
- Design algorithms for operations
- Optimize design
- Implementation of control
- Maximize inheritance
- Design associations
- Determine object representation
- Combine classes and associations into modules

a) Combine all The Three Models

The output of analysis phase are object model, dynamic model and functional model. In this phase the object designer converts actions and activities of the dynamic model and processes in the functional model into operations of the classes in the object model.

State diagrams constructed in the dynamic model provide the detail working of the object. Transition in the state diagram represents the change of states. In this phase the transitions are mapped into operations of the object. The action performed in a transition depends on both the event and the state of the object. So, the algorithm implemented on objects depends on the state of the object and the event.

An event generated by an object may represent an operation on another object. Events often occur in pairs; the first event triggers an action and the second event returns the result or indicates the completion of the action.

Space for Learners:

Actions initiated by a transition in a state diagram may expand into a full-fledged data flow diagram (DFD) in the functional model. The collection of processes within the DFD represents the body of an operation. The object designer must convert the DFD into a linear sequence of steps in an algorithm.

b) Design Algorithms for Operations

Algorithms are designed for each operation in the DFD. The DFD depicts what are the operations and the algorithm explains how it is done. The algorithm designer must follow the following steps in designing algorithms for operations:

- Selection of an appropriate algorithm
- Selection of an appropriate data structure
- Adding new classes and operations as necessary
- Assigning appropriate responsibility to classes

In choosing an appropriate algorithm the following criteria need to be addressed:

Computational complexity: Determine the complexity of the algorithm and choose an algorithm, which takes lesser computation time.

Ease of implementation: We need to select a simpler algorithm for implementation.

Flexibility: The algorithm designed should be flexible and there should be provision for future extension. If an algorithm is highly optimized then it is very difficult to understand and modify.

Choose an appropriate data structure: Every algorithm uses some data structures, to make an algorithm efficient, an appropriate and cost-effective data structure should be chosen.

Add new classes and operations as necessary: To hold intermediate results, we may need to add new classes. A complex operation can be decomposed into many low-level operations.

Assign appropriate responsibility to classes: Most of the operations have an obvious target but there are some operations, which can be placed at many places. This problem gets more complicated if the operation involves many objects. So, we need to take care of this issue.

Space for Learners:

c) Optimize The Design

The analysis model defines the logical information about the system, while the design model adds details about the information accesses. The system designer must make a balance between efficiency and clarity of the system.

The system designer can optimize the design by:

- Adding redundant associations to minimize access cost and maximize convenience
- Rearranging the computation for greater efficiency
- Saving the derived attributes to avoid re-computation of complicated expressions

Adding Redundant Associations for efficient access: During design, the structure of the object model is evaluated for an implementation. However, there may be situation where the associations found useful in analysis phase may not be useful in design phase. On the other hand, there may be some association which is defined for some other objects and may need to be redefined again for some other objects to simplify implementation.

Rearranging Execution Order for Efficiency: After adjustment of the structure of object model for optimization, the next thing is the optimization of the algorithm. For optimization of algorithms, we need to find out the unnecessary codes and codes that does not lead to solutions and remove those codes.

Saving Derived Attributes to Avoid Re-computation: There may exist some data which can be derived from other data. This kind of data may be termed as redundant data. Computation time can be saved if we can eliminate the redundant calculations by using previously calculated data.

d) Implementation of Control

As part of the system design, the designer must implement state diagram designed in the dynamic model. There are three basic approaches to implement it:

- To use location within the program to hold state.
- Direct implementation of state machine mechanism
- Using concurrent tasks

__Space for Learners:__

The state diagram can be converted to code as follows:

1. Identify the main path in the diagram that leads to execution of events. Identify the names of states in the path.
2. Identify alternate paths, which branch off the main path. This becomes the conditional statements in the program.
3. To identify the loops, find out backward paths that branch off the main path. Multiple backward paths become nested loops in the programs.
4. The states and conditions correspond to exceptional conditions need to be handled through exception handling or by error subroutines.

Direct implementation of state machine mechanism: The direct approach to implement control is to have a state machine engine class that keeps track of execution of states and actions. Each object instance maintains its own independent variables. The basic flow of control can be traced by creating stubs of the action routines. A stub contains the minimal piece of information regarding functions or subroutines.

e) Maximize Inheritance

The specialization and generalization relationships are both reciprocal and hierarchical. Specialization is just the other side of the generalization. The definition of classes and operations can be adjusted to make the inheritance as large as possible. It can be done in the following ways.

- Rearranging and adjustment of classes and operations to increase inheritance.
- Abstracting common behaviour out of the group of classes.
- Use of delegation to share behaviour when inheritance is semantically invalid.

Rearranging and adjustment of classes and operations to increase inheritance

Inheritance leads to reusability of already existing functionality. Inheriting more functions increases the level of reusability. However, larger inheritance requires, functions definitions to be redefined to serve multiple classes. The following adjustments can be used to increase the level of inheritance.

- Some operations may have fewer arguments than others. The missing arguments can be supplied in the function definition and they may be ignored where they are not necessary.
- Similar attributes in different classes may have different names. These attributes may be moved to a common ancestor class.
- Some operations may have fewer arguments because they are special cases of more general arguments. These special operations can be implemented by calling the general operations with appropriate parameters
- An operation may be defined on several different classes in a group but may not be required in other classes. In that case the operation can be defined as a common ancestor class and can be declared as no-operation on the classes where they are not required.

Space for Learners:

f) Design Associations

During the object design phase, all the associations in the object model are implemented. To make a decision for implementation, all the associations need to be analysed as they are used.

Analysing Association Traversal: Associations may be traversed either unidirectional or bi-directional. The unidirectional associations traversed in forward direction only and they are easy to implement. Bi-directional associations allow reverse and forward traversal, so bi-directional associations are always preferred over unidirectional associations so that we can add new behaviour or expand or modify the application rapidly.

One-way Associations: A unidirectional association can be implemented as a pointer.

Two-way Associations: There are three approaches to implement bi-directional associations as discussed below:

- Implement as an attribute in one direction only and perform a search when a backward traversal is required.
- Implement as attributes in both the directions. This results in fast access but if either attribute is updated then the other attribute must also be updated to keep the link consistent.

- Implement as a distinct association object, independent of either class. An association object is a set of pairs of associated objects stored in a single variable size object. For efficiency, an association object can be implemented using two dictionary objects, one for the forward direction and one for the backward direction.

Link Attributes: If an association has link attributes, then its implementation depends on the multiplicity. If the association is one-to-one, the link attributes can be stored as attribute of either object. If the association is many-to-one, the link attributes can be stored as attributes of the many object. If the association is many to many, the link attributes cannot be associated with either of the objects.

g) Determine Object Representation

The object designer has to choose when to use primitive types in representing the objects or when to combine the groups of objects. A class can be defined in terms of other classes but ultimately all data members have to be defined in terms of built-in data types supported by a programming language.

h) Packaging of Classes and Associations into Modules

Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules. Modules serve as the physical containers in which we declare the classes and objects. A module can be edited, compiled or imported separately. Different object-oriented programming languages support the packing in different ways. For example, Java supports in the form of package, C++ in the form of header files etc.
Following purposes can be solved by modularity.

- A module typically groups a set of class definitions and objects to implement some service or abstraction.
- A module is frequently a unit of division of responsibility within a programming team. A module provides an independent naming environment that is separate from other modules within the program.
- A Modules support team engineering by providing isolated name spaces.

Packaging involves the following three issues:

- Information Hiding

- Coherence of Entities
- Constructing Physical Modules

Information Hiding: During analysis phase we are not concerned with information hiding. So, visibilities of class members are not specified during analysis phase. It is done during object design phase. In a class, data members and internal operations should be hidden, so, they should be specified as private. External operations form the interface so they should be specified as public.

Coherence of Entities: Module, class, method etc. are entities. An entity is said to coherent, if it is organized on a consistent plan and all its parts fit together toward a common goal. Policy needs to be designed to make the modules more coherent.

Constructing Physical Modules: Modules of analysis phase have changed as more classes and associations have been added during object design phase. The object designer has to create modules with well-defined and minimal interfaces. The classes in a module should have similar kind of things in the system. There should be cohesiveness or unity of the purpose in a module. So that the classes, which are strongly associated can be put into a single module.

CHECK YOUR PROGRESS

1. Choose the incorrect statement in terms of Objects.
   a) Objects are abstractions of real-world
   b) Objects can't manage themselves
   c) Objects encapsulate state and representation information
   d) All of the mentioned
2. Which of the following points related to Object-oriented development (OOD) is true?
   a) OOA is concerned with developing an object model of the application domain
   b) OOD is concerned with developing an object-oriented system model to implement requirements
   c) All of the mentioned
   d) None of the mentioned

Space for Learners:

Space for Learners:

3. Which of the following is a disadvantage of OOD ?
   a) Easier maintenance
   b) Objects may be understood as stand-alone entities
   c) Objects are potentially reusable components
   d) None of the mentioned

4.Inherited object classes are self-contained.
   a) True
   b) False

3.9 SUMMING UP

- A model is a simplified representation of reality. It provides a means for conceptualization and communication of ideas in a precise and unambiguous form.
- Object Oriented Methodology is a new system development approach encouraging and facilitating reuse of software components.
- The basic steps of system designing using object-oriented methodology includes analysis, system design, object design and implementation.
- The object model describes the static, structural and data aspects of a system.
- The dynamic model describes the temporal, behavioural and control aspects of a system.
- The functional model describes the transformational and functional aspects of a system.
- Every system has all the three models. Each model describes one aspect of the system but at the same time contains references to the other models.
- An object is a concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand.
- An object has the following four main characteristics - unique identification, set of attributes, set of states, and set of operations (behaviour).
- Class is a template where certain basic characteristics of a set of objects are defined. A class defines the basic attributes and the operations of the objects of that type.

Space for Learners:

- A link is a physical or conceptual connection between object instances.
- Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. The system design process is generally divided into two subphases – logical design and physical design.
- The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system.
- The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is input into a system, how it is verified/authenticated, how it is processed, and how it is displayed as output.
- Object oriented design is a process of refinement or adding details. The object-designer works to implement the objects discovered during analysis phase

3.10 ANSWERS TO CHECK YOUR PROGRESS

1. b) Objects can't manage themselves
2. c) All the mentioned
3. d) None of the mentioned
4. b) False

3.11 POSSIBLE QUESTIONS

Short answer type questions:

1. What is model? Why do we model?
2. What is object? Discuss the main characteristics of the object with examples from the real world.
3. What is class? Discuss the relationships between class and object.
4. Define association.

Long answer type questions:

1. What is object-oriented methodology? What are the advantages of object-oriented methodology?

Space for Learners:

2. What is object-oriented process? Discuss the steps of object-oriented process.
3. What are the three models involved in object-oriented Analysis? Define each one of them.
4. What do you mean by object design? What are the steps followed during the object design?
5. How can you combine object model, dynamic model and functional model to obtain operations on classes?
6. What are the steps an object designer has to follow during algorithm design?
7. What is dynamic model? How is it represented?
8. What is system design? What are two types of system design? Explain.

3.12 REFERENCES AND SUGGESTED READINGS

1. Object-Oriented Modeling and Design with UML, M. Blaha, J. Rumbaugh, Pearson Education
2. Object-Oriented Analysis & Design with the Unified Process, Satzinger, Jackson, Burd, Thomson
3. Object Oriented Analysis & Design, Grady Booch, Addison Wesley
4. Timothy C. Lethbridge, Robert Laganiere, Object Oriented Software Engineering, TMH