1.0 Introduction

Most of the methods used in software development houses are based on a functional and/or data-driven decomposition of the systems. These approaches differ in many ways from the approaches taken by object-oriented methods where data and functions are highly integrated. Object-oriented systems development is a way to develop software by building self-contained modules or objects that can be easily replaced, modified and reused. It depicts the view of real world as a system of cooperative and collaborative objects. In this, software is a collection of discrete objects that encapsulates data and operations performed on that data to model real world objects. A class describes a group of objects having similar structures and similar operations.

Object Oriented Philosophy is very much similar to real world and hence is gaining popularity as the systems here are seen as a set of interacting objects as in the real world. To implement this concept, the process-based structural programming is not used; instead objects are created using data structures. Just as every programming language provides various data types and various variables of that type can be created, similarly, in case of objects certain data types are predefined.

There are several object-oriented development methods around. In this chapter, Object Modeling Technique (OMT) will be discussed. The OMT method given by Rumbaugh et. al. is based on entity/relationship modeling with extensions to modeling classes, inheritance and behavior.

1.1 Objectives

The main objective of this chapter is to introduce the concept of modeling in general and to give an overview of object-oriented methodologies. This chapter introduces OMT methodology and its three basic models viz object

model, dynamic model and functional model. Concepts of objects and classes are discussed in detail along with the notations of OMT methodology. Links and associations are defined and illustrated with lucid examples.

1.2 Introduction to Modeling

1.2.1 What is modeling?

A model is an abstraction of something for the purpose of understanding it before building it. Because, real systems that we want to study are generally very complex. In order to understand the real system, we have to simplify the system. So a model is an abstraction that hides the non-essential characteristics of a system and highlights those characteristics, which are pertinent to understand it. Efraim Turban describes a model as a simplified representation of reality. A model provides a means for conceptualization and communication of ideas in a precise and unambiguous form. The characteristics of simplification and representation are difficult to achieve in the real world, since they frequently contradict each other. Thus modeling enables us to cope with the complexity of a system.

Most modeling techniques used for analysis and design involve graphic languages. These graphic languages are made up of sets of symbols. As you know one small line is worth thousand words. So, the symbols are used according to certain rules of methodology for communicating the complex relationships of information more clearly than descriptive text.

Modeling is used frequently, during many of the phases of the software life cycle such as analysis, design and implementation. Modeling like any other object-oriented development, is an iterative process. As the model progresses from analysis to implementation, more detail is added to it.

1.2.2 Why do we model?

Before constructing anything, a designer first build a model. The main reasons for constructing models include:

- To test a physical entity before actually building it.
- To set the stage for communication between customers and developers.
- For visualization i.e. for finding alternative representations.
- For reduction of complexity in order to understand it.

1.3 Object Oriented Methodologies

We live in a world of objects. These objects exist in nature, in man-made entities, in business, and in the products that we use. They can be categorized, described, organized, combined, manipulated and created. Therefore, an object-oriented view has come into picture for creation of computer software. An object-oriented approach to the development of software was proposed in late 1960s.

Object Oriented Methodology (OOM) is a new system development approach encouraging and facilitating reuse of software components. With this methodology, a computer system can be developed on a component basis, which enables the effective reuse of existing components and facilitates the sharing of its components by other systems. By using OOM, higher productivity, lower maintenance cost and better quality can be achieved.

OOM requires that object-oriented techniques be used during the analysis, design and implementation of the system. This methodology makes the analyst to determine what the objects of the system are, how they behave over time or in response to events, and what responsibilities and relationships an object has to other objects. Object-oriented analysis has the analyst look at all the objects in a system, their commonalties, difference, and how the system needs to manipulate the objects.

During design, overall architecture of the system is described. During implementation phase, the class objects and the interrelationships of these classes are translated and actually coded using the programming language. The databases are created and the complete system is made operational.

1.3.1 Object Oriented Process

The OOM for building systems takes the objects as the basis. For this, first the system to be developed is observed and analyzed and the requirements are defined. Once this is done, the objects in the required system are identified. For example, in case of a Banking System, a customer is an object, a ledger is an object, passbook is an object and even an account is an object.

OOM is somewhat similar to the traditional approach of system designing, in that it also follows a sequential process of system designing but with a different approach. The basic steps of system designing using OOM may be listed as:

- System Analysis
- System Design
- Object Design
- Implementation

1.3.1.1 System Analysis

As in any other system development model, system analysis is the first phase of OOM too. In this phase, the developer interacts with the user of the system to find out the user requirements and analyses the system to understand the functioning of it.

Based on this system study, the analyst prepares a model of the desired system. This model is purely based on what the system is required to do. At this stage the implementation details are not taken care of. Only the model of the system is prepared based on the idea that the system is made up of a set of interacting objects. The important elements of the system are emphasized.

1.3.1.2 System Design

System Design is the next development stage in OOM where the overall architecture of the desired system is decided. The system is organized as a set of sub systems interacting with each other. While designing the system as a set of interacting subsystems, the analyst takes care of specifications as observed in system analysis as well as what is required out of the new system by the end user.

The system analysis is to perceive the system as a set of interacting objects. A bigger system may also be seen as a set of interacting smaller subsystems that in turn are composed of a set of interacting objects. While designing the system, the stress lies on the objects comprising the system and not on the processes being carried out in the system.

1.3.1.3 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. Here the implementation of these objects is decided in the form of data structures required and the interrelationships between the objects. For example, we can define a data type called customer and then create and use several objects of this data type. This concept is known as creating a class.

In this phase of the development process, the designer also decides about the classes in the system based on these concepts. He decides on whether the classes need to be created from scratch or any existing classes can be used as it is or new classes can be inherited from them.

1.3.1.4 Implementation

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

1.3.2 Advantages of Object Oriented Methodology

- As compared to the conventional system development techniques, OOM provides many benefits.
- The systems designed using OOM are closer to the real world as the real world functioning of the system is directly mapped into the system designed using OOM. Because of this, it becomes easier to produce and understand designs.
- The objects in the system are immune to requirement changes because of data hiding and encapsulation features of object-orientation. Here, encapsulation we mean a technique that allows the programmer to hide the internal functioning of the objects from the users of the objects. Encapsulation separates the internal functioning of the object from the external functioning thus providing the user flexibility to change the external behavior of the object making the programmer code safe against the changes made by the user.
- OOM designs encourage more reusability. The classes once defined can easily be used by other applications. This is achieved by defining classes and putting them into a library of classes where all the classes are maintained for future use. Whenever a new class is needed the programmer first looks into the library of classes and if it is available, it can be used as it is or with some modification. This reduces the development cost & time and increases quality.
- Another way of reusability is provided by the inheritance feature of the object-orientation. The concept of inheritance allows the programmer to use the existing classes in new applications i.e. by making small additions to the existing classes can quickly create new classes. This provides all the benefits of reusability discussed in the previous point.
- As the programmer has to spend less time and effort so he can utilize saved time (due to the reusability feature of the OOM) in concentrating on other aspects of the system.

- OOM approach is more natural as it deals with the real world objects. So, it provides nice structures for thinking and abstracting and leads to modular design.

Many OOMs have been developed since its inception. Some of the popular object oriented methodologies are listed below:

- Booch Methodology [1994]: He developed the Object Oriented Analysis and Object Oriented Design (OOA/OOD) concepts.
- RDD Methodology [1990]: Wirfs-Brock, Wilkerson, and Wiener developed Responsibility Driven Design (RDD) methodology.
- OMT methodology [1991]: James Rumbaugh led a team at research labs of General Electric to develop the Object Modeling Technique (OMT).
- OOSE [1994]: Ivar Jacobson developed the Object Oriented Software Engineering (OOSE).

Other Object-Oriented methodologies that have been around in the world are:

- Berand [Berand 93]
- BON [Nerson 92]
- Coad/Yourdon [Coad 1991]
- Embley [Embley 1993]
- EVB [Jurik 1992]
- ROOM [Selic 1994]
- FUSION [Coleman 1994]
- HOOD [HOOD 89]
- Shlaer and Mellor (Shlaer 1992]

Discussion of these methodologies is beyond the scope of this course material as these are not the part of your course curriculum. Object Modeling Technique (OMT) methodology [1991] developed by James Rumbaugh et. al.

is the part of your syllabus and will be discussed in detail in rest of the chapters.

1.4 OMT Methodology

OMT is an object-oriented software development methodology given by James Rumbaugh et.al. This methodology describes a method for analysis, design and implementation of a system using object-oriented technique. It is a fast, intuitive approach for identifying and modeling all the objects making a system. The OMT consists of three related but different viewpoints each capturing important aspects of the system i.e. the static, dynamic and functional behaviors of the system. These are described by object model, dynamic model and functional model of the OMT.

The object model describes the static, structural and data aspects of a system. The dynamic model describes the temporal, behavioral and control aspects of a system. The functional model describes the transformational and functional aspects of a system. So every system has these three aspects. Each model describes one aspect of the system but contains references to the other models.

The entire OMT software development process has four phases: analysis, system design, object design, and implementation of the software. Most of the modeling is performed in the analysis phase. In this phase, three basic models - Object Model, Dynamic Model and Functional Model are developed. While the Object Model is most important of all as it describes the basic element of the system, the objects, all the three models together describe the complete functional system.

Object Model describes the objects in a system and their interrelationships. This model observes all the objects as static and does not pay any attention to their dynamic nature. Dynamic Model depicts the dynamic aspects of the system. It portrays the changes occurring in the states of various objects with the events that might occur in the system. Functional Model basically

describes the data transformations of the system. This describes the flow of data and the changes that occur to the data throughout the system.

In this chapter, we will discuss the object model in detail. Remaining models will be described in the ensuing chapters.

1.4.1 Object Model

The object model describes the structure of the objects in the system - their identity, their relationships to other objects, their attributes, and their operations. The object model depicts the primary view of how the real world in which the system interacts is divided and the overall decomposition of the system. The object model provides the framework into which the other models are placed.

The object model is represented graphically with an object diagram. The object diagram contains classes interconnected by association 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.

Now we discuss the concepts of object, class and relationships between objects and/or classes. We also introduce the notations and symbols of OMT used for object, class and relationships.

1.4.1.1 Object and Class

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. Defining a class does not define any object, but it only creates a template. For objects to be actually created, instances of the class are to be created as per the requirement of the case.

Classes are built on the basis of abstraction, where a set of similar objects is observed and their common characteristics are listed. Of all these, the characteristics of concern to the system under observation are taken and the class definition is made. The attributes of no concern to the system are left out. This is known as abstraction. So, the abstraction is the process of hiding superfluous details and highlighting pertinent details in respect to the system under development.

It should be noted that the abstraction of an object varies according to its application. For instance, while defining a pen class for a stationery shop, the attributes of concern might be the pen color, ink color, pen type etc., whereas a pen class for a manufacturing firm would be containing the other dimensions of the pen like its diameter, its shape and size etc.

Each application-domain concept from the real world that is important to the application should be modeled as an object class. Classes are arranged into hierarchies sharing common structure and behavior and are associated with other classes. This gives rise to the concept of inheritance.

Through inheritance, a new type of class can be defined using a similar existing class with a few new features. For instance, a class vehicle can be defined with the basic functionality of any vehicle and a new class called car can be derived out of it with a few modifications. This would save the developers time and effort as the classes already existing are reused without much change.

In OMT, classes are represented by a rectangular box which may be divided into three parts as shown in Figure 1.1. The top part contains the name of the class written in bold, middle part contains a list of attributes and bottom part contains a list of operations.

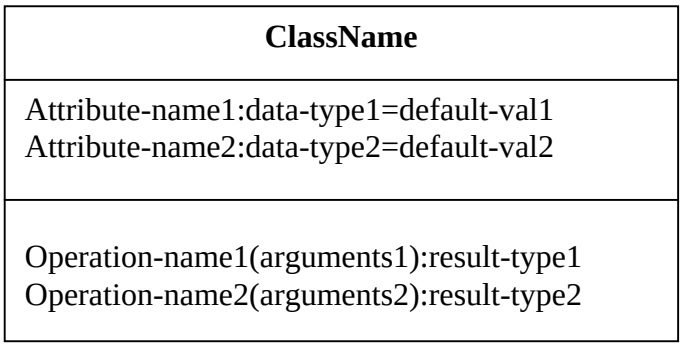


Figure 1.1

An attribute is a data value held by objects in a class. Each attribute has a value for each object instance. This value should be a pure data value, not an object. Attributes are listed in the second part of the class box. Attributes may or may not be shown; it depends on the level of detail desired. Each attribute name may be followed by the optional details such as type and default value. An object model should generally distinguish independent base attributes from dependent derived attributes. A derived attribute is that which is derived from other attributes. For example, age is a derived attribute, as it can be derived from date-of-birth and current-date attributes.

An operation is a function or transformation that may be applied to or by objects in a class. Operations are listed in the third part of the class box. Operations may or may not be shown; it depends on the level of detail desired. Each operation may be followed by optional details such as argument list and result type. The name and type of each argument may be given. An empty argument list in parentheses shows explicitly that there are no arguments. All objects in a class share the same operations. Each operation has a target object as an implicit argument. An operation may have arguments in addition to its target object, which parameterize the operation. The behavior of the operation depends on the class of its target.

An operation may be polymorphic in nature. A polymorphic operation means that the same operation takes on different forms in different/same classes. Overloading of operators, overloading of functions and overriding of functions

provided by object-oriented programming languages are all examples of polymorphic operations. A method is the implementation of an operation for a class. The method depends only on the class of the target object.

Now, we present a number of examples of classes to clarify what we have discussed above. Figure 1.2 shows the class Book. Attributes of Book are title, author, publisher along with their data types. Operations on Book are open(), close(), read().

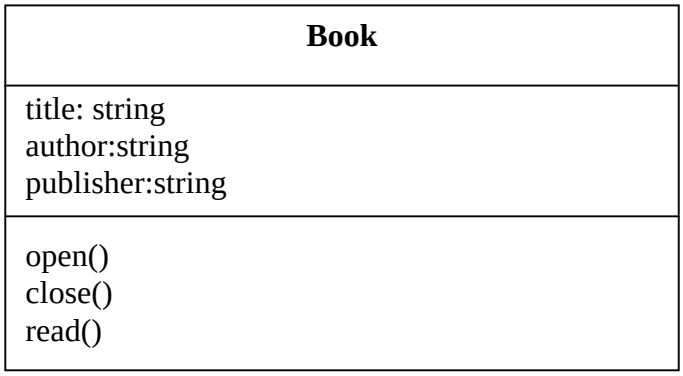


Figure 1.2

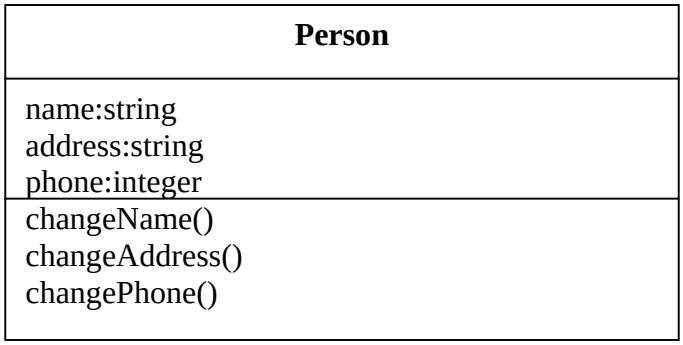


Figure 1.3

Figure 1.3 shows the class Person with name, address & phone along with their data types as attributes and changeName(), changeAddress() & changePhone() as operations.

Figure 1.4 shows the class Window with xMin, yMin, xMax, yMax along with their data types as attributes and draw(), cut(), erase() & move() as operations.

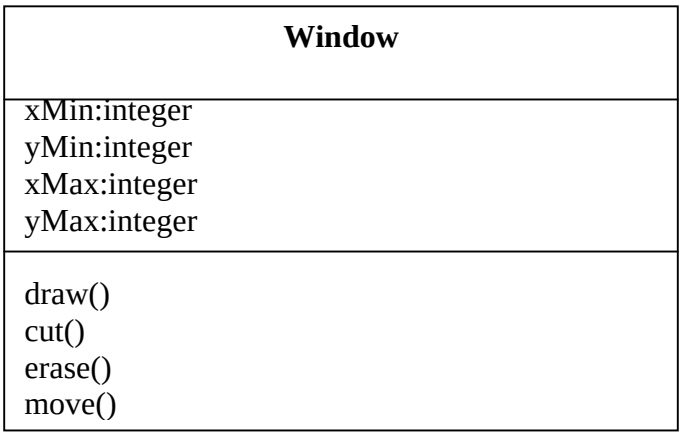


Figure 1.4

Now, let us formally define an object. 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
- Set of operations (behavior)

Unique identification, we mean every object has a unique name by which it is identified in the system. Set of attributes, we mean every object has a set of properties in which we are interested in. Set of states we mean values of attributes of an object constitute the state of the object. Every object will have a number of states but at a given time it can be in one of those states. Set of operations we mean externally visible actions an object can perform. When an operation is performed, the state of the object may change.

In other words, an object is an instance of an object class. Figure 1.2 (rounded box) represents an object instance in OMT. Object instance is a particular object from an object class. The box may/may not be divided in particular regions. Object instances can be used in instance diagrams, which are useful for documenting test cases and discussing examples.

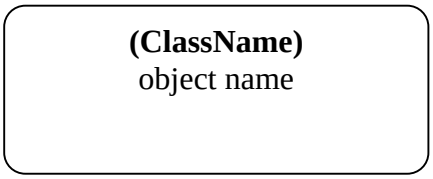


Figure 1.5

Now we give three examples of objects of class Book shown in Figures 1.2, which are shown in Figure 1.6 (a), (b) & (c) respectively.

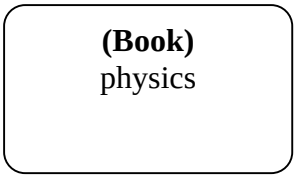


Figure 1.6(a)

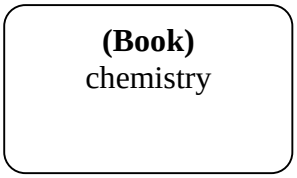


Figure 1.6(b)

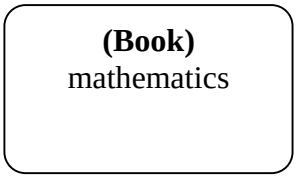


Figure 1.6(c)

Here we give three examples of objects of class Person shown in Figures 1.3, which are shown in Figure 1.7 (a), (b) & (c) respectively.

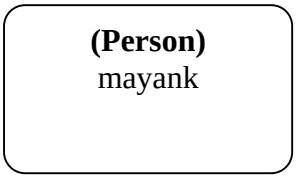


Figure 1.7(a)

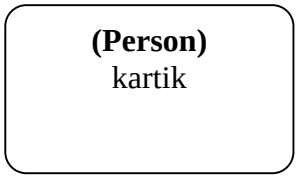


Figure 1.7(b)

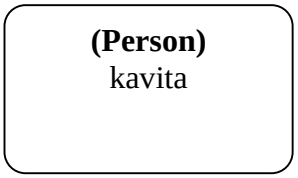
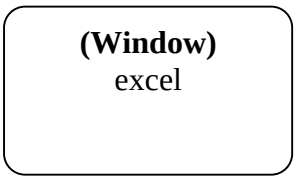
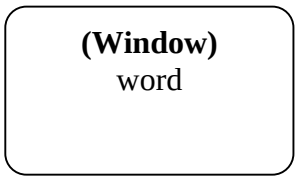


Figure 1.7(c)

Next we give three examples of objects of class Window shown in Figures 1.4, which are shown in Figure 1.8 (a), (b) & (c) respectively.





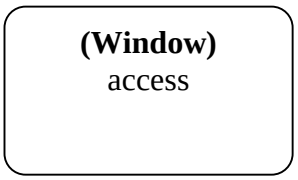


Figure 1.8(a) **Figure 1.8(b)** **Figure 1.8(c)**

Let us introduce the concept of derived object. It is defined as a function of one or more objects. It is completely determined by the other objects. Derived object is redundant but can be included in the object model.

After having discussed the concepts of objects and their classes, let us discuss relationships between objects and between classes.

1.4.1.2 Links and Associations

A link is a physical or conceptual connection between object instances. In OMT, link is represented by a line labeled with its name as shown in Figure 1.9.

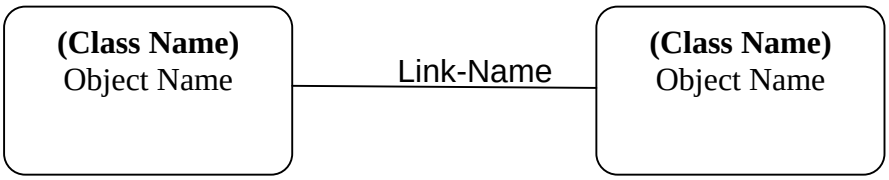


Figure 1.9

For example Mayank flies Jaguar. So ‘flies’ is a link between object instance Mayank of class Person and object instance Jaguar of class Airplane as shown in Figure 1.10. Kartik reads Physics. Here reads is a link between object instance Kartik of class Person and object instance Physics of class Book as shown in Figure 1.11.

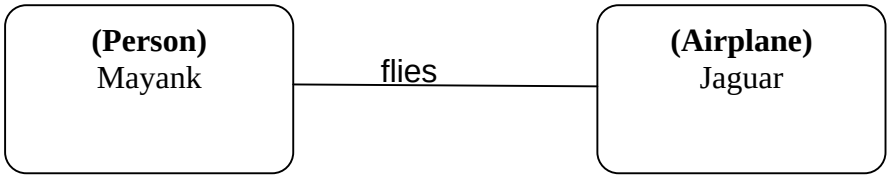


Figure 1.10

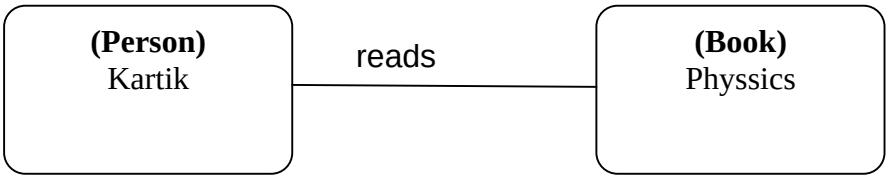


Figure 1.11

An association describes a group of links with common structure and common semantics between two or more classes. Association is represented by a line labeled with the association name in italics as shown in Figure 1.12.

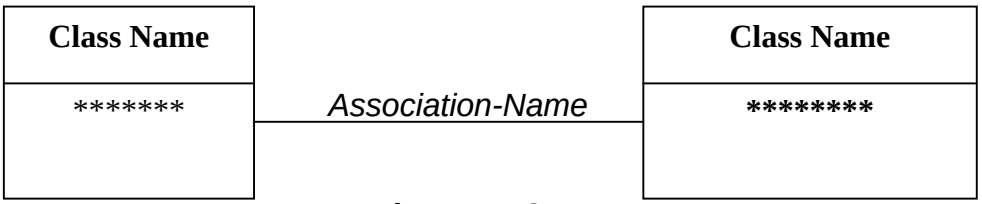


Figure 1.12

Association names are optional. If the association is given a name, it should be written above the line. Association names are italized. In case of a binary association, the name reads in a particular direction (i.e. left to right), but the binary association can be traversed in either direction. For example, a pilot flies an airplane or an airplane is flown by a pilot. All the links in an association connect objects from the same classes. Associations are bi-directional in nature.

Multiplicity: It specifies how many instances of one class may relate to a single instance of an associated class. Multiplicity constrains the number of related objects.

There are special line terminators to indicate certain common multiplicity values. A solid ball is the symbol for "many", meaning zero, one or more. A hollow ball indicates "optional", meaning zero or one. The multiplicity is indicated with special symbols at the ends of association lines. In the most general case, multiplicity can be specified with a number or set of intervals. If no multiplicity symbol is specified that means a one-to-one association. The rules of multiplicity are summarized below:

- Line without any ball indicates one-to-one association.
- Hollow ball indicates zero or one.
- Solid ball indicates zero, one or more.
- Numbers written on solid ball such as 1,2,6 indicates 1 or 2 or 6.
- Numbers written on solid ball such as 1+ indicates 1 or more, 2+ indicates 2 or more etc.

An association can be unary, binary, ternary or n-ary. Unary association is between the same class as shown in Figure 1.13.

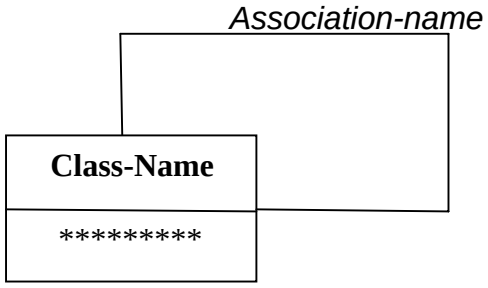


Figure 1.13

Example of unary association is Person teaches Person as shown in Figure 1.14. Other examples of unary association can be Person marries a Person, Person manages Person etc.

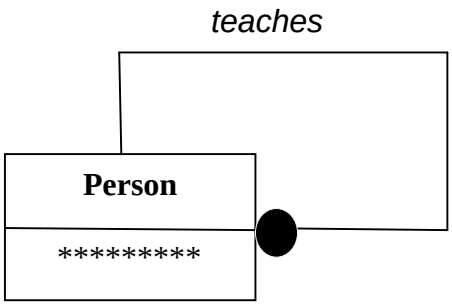


Figure 1.14

A binary association is an association between two classes as shown in Figure 1.15.

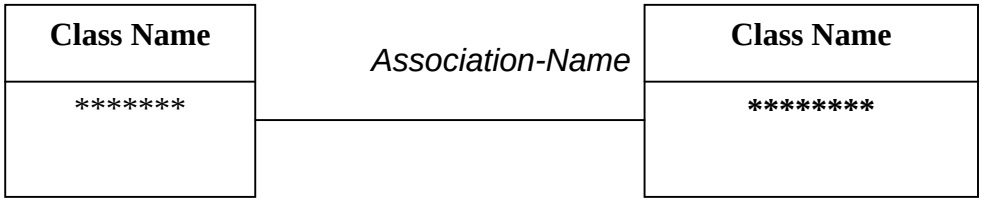


Figure 1.15

Example of a binary association is “Person sits on a Chair”. One person can sit at one chair. So multiplicity of this association is one-to-one as shown in Figure 1.16.

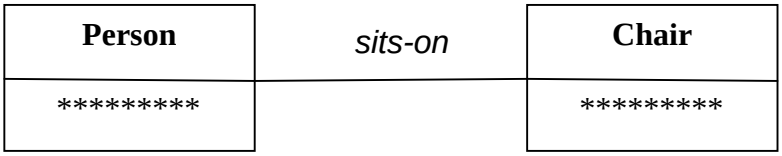


Figure 1.16

Example of a binary association is “Supplier supplies Parts”. One supplier can supply many parts or one part can be supplied by many suppliers. So multiplicity of this association is many-to-many as shown in Figure 1.17.

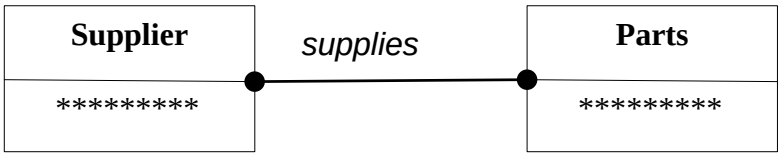


Figure 1.17

Another example of binary association is “Person possesses a Passport”. Either a person can have one passport or no passport but one passport can be with one person. So multiplicity of this association is one-to-optional as shown in Figure 1.18.

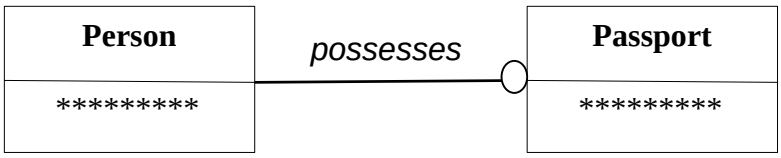


Figure 1.18

Another example of binary association is “Company employs Person”. One company can employ zero, one or more persons but one person can be employed in one company only (assume). So multiplicity of this association is one-to-many as shown in Figure 1.19.

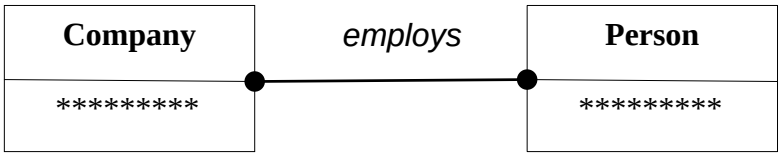


Figure 1.19

Ternary association is an association among three classes. On the same line, n-ary association is an association among n classes. The OMT symbol for ternary and n-ary associations is a diamond with lines connecting to related classes as shown in Figure 1.20. A name for the association is optional and is

written next to the diamond. An n-ary associations cannot be subdivided into binary associations without losing information.

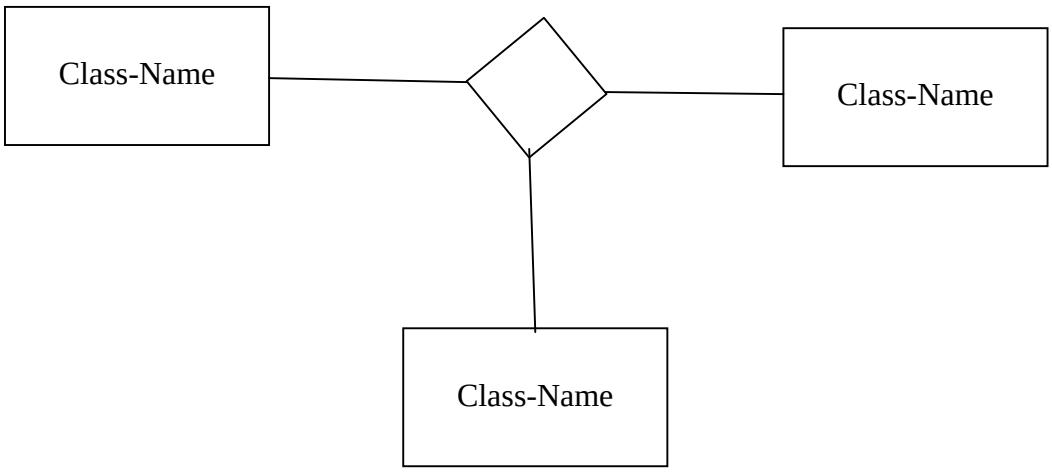


Figure 1.20

Now let us consider the example of a ternary association. Programmers develop Projects in (programming) Languages. One programmer can be engaged in zero, one or more projects and can know zero, one or languages. Similarly, one project can be developed by one or more programmers and in one or more languages. So this association along with its multiplicity is shown in Figure 1.21.

Other examples of ternary and higher order associations are “Teacher teaches Students in a Classroom”, “Doctor diagnoses Patient in Room at a given Schedule” etc.

There are many other concepts related to associations such as link attribute, link class, role names, qualifiers etc. which will be discussed in the next chapter.

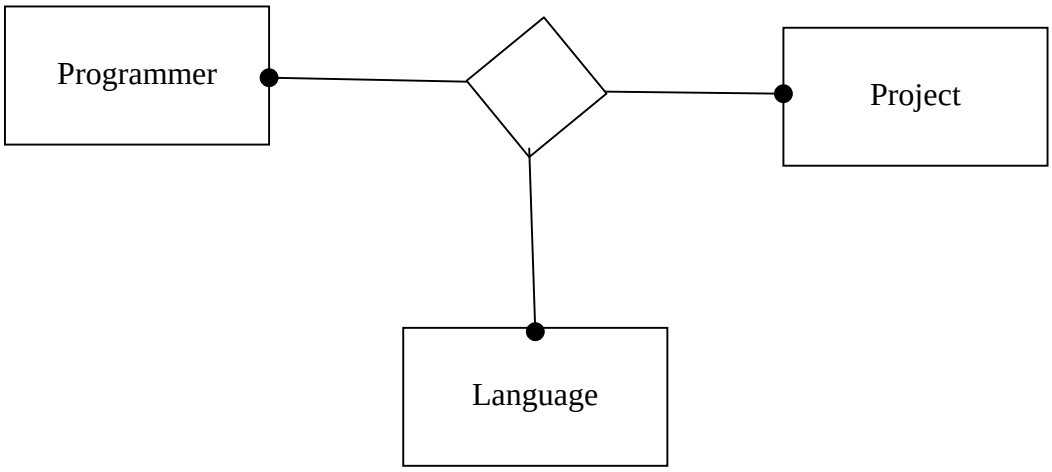


Figure 1.21

1.5 Summary

In this chapter you have learnt the concepts of model, object modeling, OMT methodology, object model, dynamic model, functional model, class, object, link and association, which are summarized below:

- 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 (OOM) is a new system development approach encouraging and facilitating reuse of software components. By using OOM, higher productivity, lower maintenance cost and better quality can be achieved.
- The basic steps of system designing using OOM include 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, behavioral 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 (behavior).
- 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.
- Defining a class does not define any object, but it only creates a template. For objects to be actually created, instances of the class are to be created as per the requirement of the case.
- A link is a physical or conceptual connection between object instances. In OMT, link is represented by a line labeled with its name.
- An association describes a group of links with common structure and common semantics between two or more classes. Association is represented by a line labeled with the association name in italics.
- An association may be unary, binary, ternary or n-ary.
- Multiplicity specifies how many instances of one class may relate to a single instance of an associated class.

1.6 Suggested Readings/Reference Materials

1. Object-Oriented Modeling and Design with UML, M. Blaha, J. Rumbaugh, Pearson Education-2007
2. Object-Oriented Analysis & Design with the Unified Process, Satzinger, Jackson, Burd, Thomson-2007
3. Object Oriented Analysis & Design, Grady Booch, Addison Wesley-1994

4. Timothy C. Lethbridge, Robert Laganiere, Object Oriented Software Engineering, TMH, 2004

1.7 Self-Assessment Questions

1. What is model? Why do we model?
2. What is object oriented methodology? What are the advantages of object oriented methodology?
3. What is object oriented process? Discuss the steps of object oriented process.
4. What is Object Modeling Technique (OMT)? What are the phases of OMT? Discuss each in brief.
5. What are the three models involved in OMT? Define each one of them.
6. What is object? Discuss the main characteristics of the object with examples from the real world.
7. List 20 objects from the real world around you. Write their attributes and operations.
8. What is class? What is OMT notation for a class Discuss the relationships between class and object.
9. List 10 classes from the real world and define them.
10. Define link and give five different examples of links and represent them using OMT notations.
11. Define association. Discuss the characteristics and OMT notations of the association.
12. Give five examples of each unary, binary and ternary associations from the real world.
13. What do you mean by multiplicity of the association? Discuss different types of multiplicity by giving suitable examples.