
UNIT 2 STRUCTURAL MODELING USING UML

Structure	Page no.
2.0	Introduction
2.1	Objectives
2.2	Introduction to UML
2.3	Basic Structural Modeling
2.3.1	Classes
2.3.2	Relationships
2.3.3	Common Mechanisms
2.3.4	Class Diagram
2.4	Advanced Structural Modeling
2.5	Advanced Classes
2.6	Advanced Relation
2.7	Interfaces Type and Roles
2.7.1	Package
2.7.2	Instance and Object Diagrams
2.8	Summary
2.9	Solutions/Answers to check your progress
2.10	References/Further Reading

2.0 INTRODUCTION

As you know the software application development process consists of many stages. For the development of enterprise applications, there is a team such as Analyst, designer, coder, tester, technical writer and each member of the team is worked on a different stage and needs a different level of detail. For example, the coder needs a design document for converting into coding, and the technical writer wants to know about the entire system's behaviour. The UML provides a communicative solution to all team members so that they can take help from at least a solitary UML diagram. This Unit introduces you to UML and its different diagrams, which are used in designing the systems and help in building a model for overall system development.

In the previous Unit of this block, you have already learned the basic principles and constructs in object orientation mechanism. This Unit explains you how to construct the class diagram. Before defining a class diagram, in this Unit explains about the basic concepts classes, relationships and some important terms or notations that are useful in designing UML diagrams. This Unit will also introduce you to some advanced concepts related to structuring modelings, such as advanced classes and relations. In addition, this Unit also discuss the type or stereotype of class which is generally used in the analysis process to identify the objects for a system. At the end of this Unit, you will be introduced to some other UML diagrams like package and object diagrams.

2.1 OBJECTIVES

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

- Explain about UML and its advantage,
- Describe different types of modeling ,
- Explain classes and their relationships,
- Describe how to design class diagram,
- To draw package diagram, and
- To draw object diagram

2.2 INTRODUCTION TO UML

Modeling is a tested and well-known technique that helps build a model. Model is a blueprint of the real system that needs to be built. You can say that modeling is a necessity for designing software systems. The vocabulary and rules of modeling language is used to create a conceptual and physical illustration for any software system. UML (Unified Modeling Language) is a standard modeling language, and it is different from the other programming languages such as C++, Java. It is a pictorial language used to build models for software systems. UML is used to design models for large, medium as well as small applications. UML was created by the Object Management Group (OMG), and the current version of the Unified Modeling Language™ is UML 2.5.1, released in June 2017. For designing applications, you can use UML development tool. Many tools like StarUML, ArgoUML, EclipseUML and so on, are used for UML diagram design/drawing.

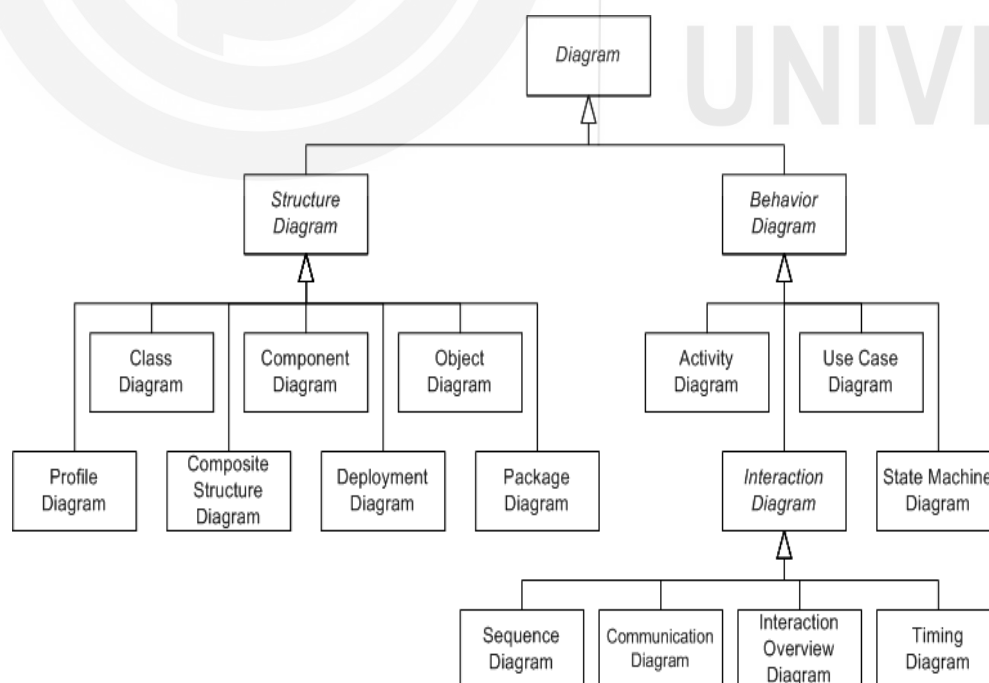


Figure 2.1: Hierarchy of UML diagrams

Basically UML diagrams denote two different views of a system model as static and dynamic. The static or structural view focuses on the system's static structure using objects, attributes, operations, and relationships. Structural diagrams do not employ time-connected concepts and do not display the details of dynamic behaviour. There are seven types of structural diagram such as class diagram, object diagram, component diagram and composite structure diagram. The dynamic or behavioral view represents the dynamic nature of the system, and this view contains diagrams like use case diagram, interaction diagram, activity diagram and state machine diagrams. The hierarchy of UML diagrams is given in figure-2.1.

This Unit covers basic as well as advanced structural modeling. There are three types of modeling in UML such as Structural modeling, Behavioral modeling and Architectural modeling. The Structural modeling describes the static features of a system. The behavioral modeling defines the interaction within the system. The architectural modeling represents the complete framework of the system; it means that it contains both structural and behavioral elements of the system. Architectural model can be expressed as the blueprint of the entire system. You can find more about the behavioral modeling in Unit 3 and 4. The architectural modeling and their related diagrams such as component, collaboration and deployment diagram will be defined in Unit 5 of this Block.

There are many advantages of using UML, which are as follows:

- You may specify the structure and behavior of the system, which actually shows the blueprint of the system.
- It breaks the large complex system into separate fragments that can be understood easily.
- It can be easily grasped by the new team members.
- The Designed UML model is not platform-specific. Hence it can be used by any developers who are working on different platforms.

UML notations play a very important role in building a complete and meaningful model. When a model is demonstrated properly, then only its purpose is completely fruitful; otherwise, it is useless. In subsequent sections, you will find about these UML notations with other elements descriptions as you read further.

2.3 BASIC STRUCTURAL MODELING

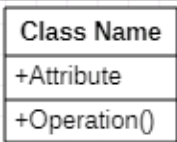
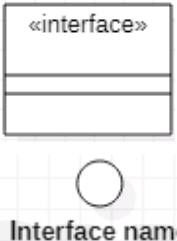

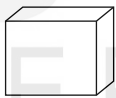
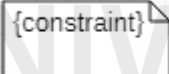
This section introduces the structural modeling through which you can understand the structure of the system and draw a structural model like a class diagram. This section comprises four subsections which emphasize on classes, relationships, class diagram and common mechanisms.

Modeling means creating a diagram for a system that includes identifying the elements that are important to your particular module. The structural modeling gives a structural view of a system that highlights the structure of the objects including their classifiers, relationships, attributes and operations. These elements form the vocabulary of the system you are modeling. For example, if you are going to buy a car, things like wheels, frame size, color, lights, and engine are some things that will be worthful to you in understanding about the car; these things are properties of the car. In UML, all of these things are modelled as classes. Cars have external and internal structure, only important properties of car are visible and the rest are hidden. This feature is called abstraction. A class is an abstraction of the things that are a part of your vocabulary. For creating a structural model, you have to collect key data contained in the problem domain.

In the previous Unit, you have already got awareness about the basic principles and constructs in object orientation paradigms which are helpful for understanding the concepts of structural modeling. The class diagram is one of the most important structural model in UML.

The core elements of the structural modeling are class, interface, component, node and constraints and they are described with their notations in Table 2.1.

Table 2.1: Core elements of the structural modeling

Construct	Description	Syntax
Class	Class is a 'blueprint' for creating objects. It contains a group of objects that share the same properties, operations, and relationships. It contains three sections as name, attributes and operations.	
Interface	It is a collection of named operations that describe the behaviour of an element. You can neither define any attributes nor any implementation of the operations in the interface.	
Component	A component diagram is used to split the actual system into various modular parts. Each component has one clear aim within the whole system.	
Node	It is a run-time physical object that denotes a computational resource such as server, router.	
Constraint	It is a condition for a UML element	

In the next sub-section, you will learn about the classes. But before going to this topic, you may know about the object which is a basic element of object oriented concepts. The object is fundamental construct which consist of data structure as well as behaviour in a single entity. For example: Mobile, TV, Ram and Car. Every object has identity like Person has name, age and address.

2.3.1 Classes

A class describes a group of objects which have common attributes or properties, operations or behavior, relationships and semantics. Classes are an essential structure of any object-oriented system.

When you develop any system, you need classes to collect the system's vocabulary. These classes may relate to the problem domain as well as implementation. Classes can be used to represent anything from hardware, software to purely conceptual things. A class describes only *important attributes to an application and ignores the rest* ; it is known as an abstraction, an important feature of object-oriented concepts.

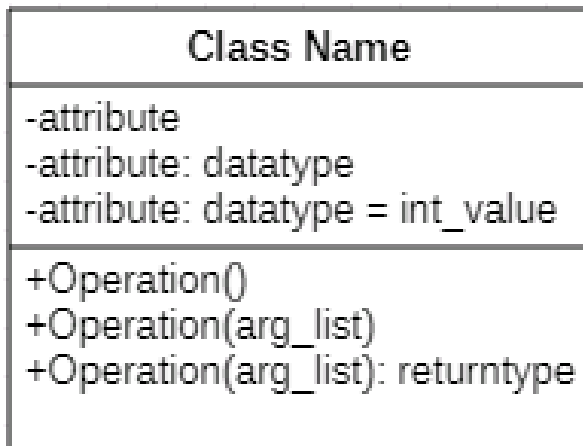


Figure 2.2: Class syntax

A class is graphically designed as a rectangle with three sections that comprises its name, attributes and operations as shown in the figure-2.2. For example, Car class contains attributes (maker name, wheel size, gears, frame size and color) and operations (repair, move) as shown in figure-2.3.

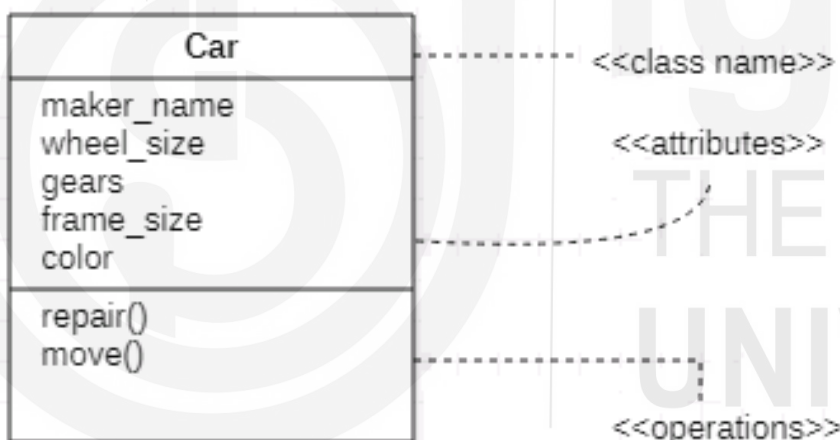


Figure 2.3: Car class with attributes and operations

You can say that a class contains infinite set of individual objects with the same data structure (attributes) and behavior (operations). Each object is termed as an instance of its class. So, each instance (known as object) of the class has its own value for each attribute but shares the attribute names and operations with other class instances.

Following are some points for creating a class:

- Each class must have a unique **name**. A class name is a textual string that comprises any numbers of letters, numeric and punctuation marks. For example, Car, Book, Teacher, Account or Person etc.
- A class may or may not have **attributes**. An attribute is defined by a name that contains a series of values. For example: Person class have attributes like name, age, address. In the similar way, Car class may have attributes like frame size, wheel size, gears, color and make as shown in figure-2.3. You can also mention attributes 'types' such as String or integer etc., as shown in figure-2.4.

- A class may or may not have **operations**. An operation describes the behavior of the class. An operation has a name like a class name. Usually, the operation name is written as a short text string that represents some behaviour of its enclosing class. For example, Car class may have operations like move, repair as shown in figure-2.4. You can identify an operation by declaring its signature. Graphically, operations are given in a section just below the class attributes.
- You can also define the visibility of class members (attributes and operations) by using plus sign (+) for public, negative sign (-) for private, hash sign (#) for protected and tilde (~) for the package. You can see in figure-2.4 where all attributes and operations are mentioned as public.

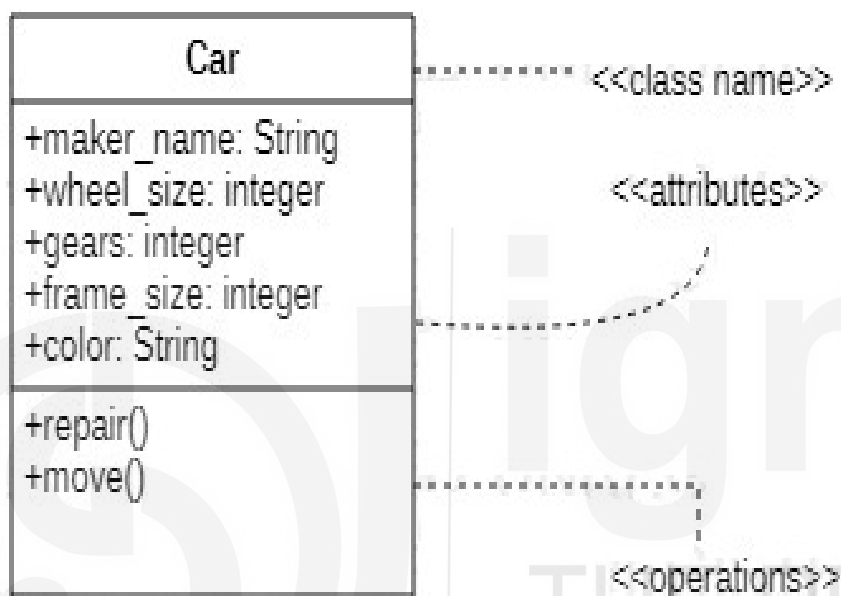


Figure 2.4 : Class attributes and visibility

2.3.2 Relationships

Relationships define how classes communicate with each other. There are three basic types of relationships in UML- Dependencies, Generalization and Associations for connecting or communicating classes to each other. Graphically, a relationship is shown as a path with different types of lines used to differentiate the categories of relationships between classes.

Dependency describes the relationships among classes in which one class is dependent on another class. It means that if you make changes in one class, it will affect the other class. For example, class Vehicle uses the properties and operations of another class (say Bodyshop) but not necessarily the reverse. You can show dependency as a dashed directed line, directed to the class being dependent upon the class. As you can see in figure-2.5, vehicle class is dependent on 'Bodyshop' class.

Generalization is a "a-kind-of" relationship. It creates relationships between superclass(base class) and subclass, enabling inheritance of attributes and operations. It defines a base(parent) class with some properties functions. A new class will be derived from this base class and it is called child or sub class of the base class. The child class can use the properties and operations of the superclass. As you can see in figure-2.5, vehicle class is super class or base or parent class and Car, Bus are child class or sub class. The generalization is depicted as a solid line with a hollow arrow pointing to the superclass.

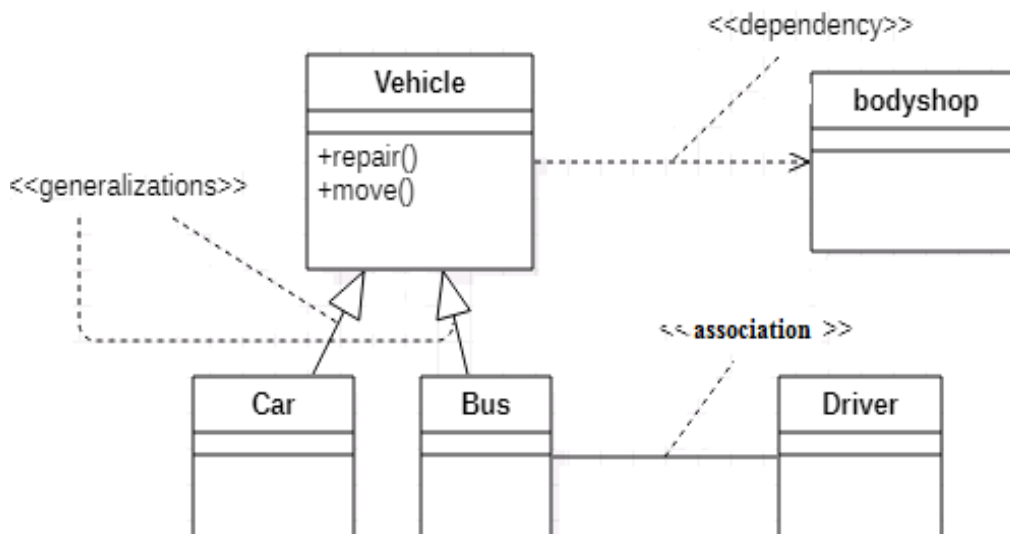


Figure 2.5: Different relationships: Dependency, Generalization and Association

Association is “a-part-of” relationships between classes. It refers to relationships between classes in which objects of one class are connected to other class objects for communicating data as you can see in figure-2.6. When you connect just two classes is called a binary association. Associations can also have more than two classes, which will be known as n-ary associations. One more form of association is called **aggregation** that specifies a whole-part relationship between the aggregate (whole) and the component part. For example, paragraph is part of word document. Each association can have a name, role, multiplicity and aggregation.

You can use a name for an association that describes the relationship's nature. You can also define role names and multiplicity for an association. Multiplicity states how many instances of one class may communicate to a single instance of other class. You can display a multiplicity of exactly one (1), zero or one (0..1), many (0..*), or one or more (1..*).



Figure 2.6: An association between classes

For example, IGNOU University have many schools. In figure-2.6, there are two classes Schools and University. Both the classes are related to each other, and their relationship is defined by association. This association has a name as Works-for, and role names are employee and employer. The employee of schools worked for employer i.e. University. Under the university, there are many schools, so multiplicity is defined as one or many.

During designing your application, you might need to describe information through representations. This section explains to you some important terms or notations that might be useful in designing UML diagrams. These notations like notes, tagged values, stereotypes, constraints are demonstrated by using the following figure-2.7:

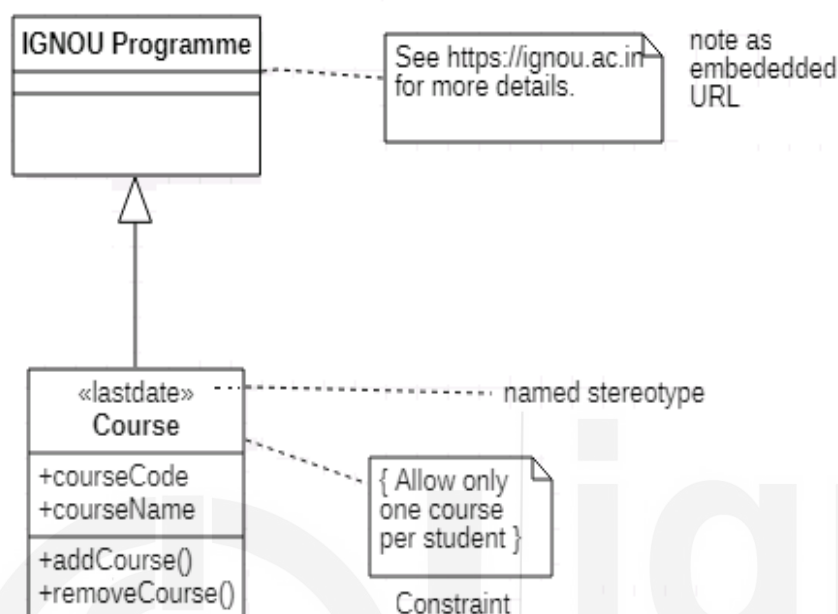


Figure 2.7: Information describing notations

Notes are used to define comments, explanations for documentation purposes only. It cannot have any effect on the semantics of the model. A note may simply comprise normal text or graphical symbols. Graphically, a note is depicted as a rectangle box with a dog-eared corner along with the comment. You can see the note as an embedded URL notation in figure-2.7.

Tagged value is used to create extension or new information of properties of UML element. Graphically, you can draw a tagged value as a string enclosed in {brackets}.

A **stereotype** is used to create new building blocks. Using stereotypes, we can use the basic elements but with special properties, semantics and notation. Graphically, you can represent stereotypes as text name inside guillemets(<< >>) and place them above the name of another element. You can see stereotype notation as a <<lastdate>> above the class course element in figure-2.7.

Constraint is defined as a condition for a UML element that will be checked at the run-time of applications. Graphically, you can draw as a string enclosed in {brackets} and can be put in a note. You can see constraint as a condition notation for course class in figure-2.7.

When you use these notations in your UML diagram, keep in mind such points as always use short notes, use notes only when the facts cannot be conversed using UML, don't use too many graphical stereotypes, tagged values and constraints; use only common and simple supplements.

2.3.4 Class Diagram

OOAD

The Class diagram is one of the most common diagram which is used to model object-oriented systems. The class diagram depicts the logical structure of the system. It shows classes, interfaces and their relationships. When you draw class diagrams, you can use only those class attributes that are important in the current context of your application and display related classes in the same diagrams. It gives the static view of the system.

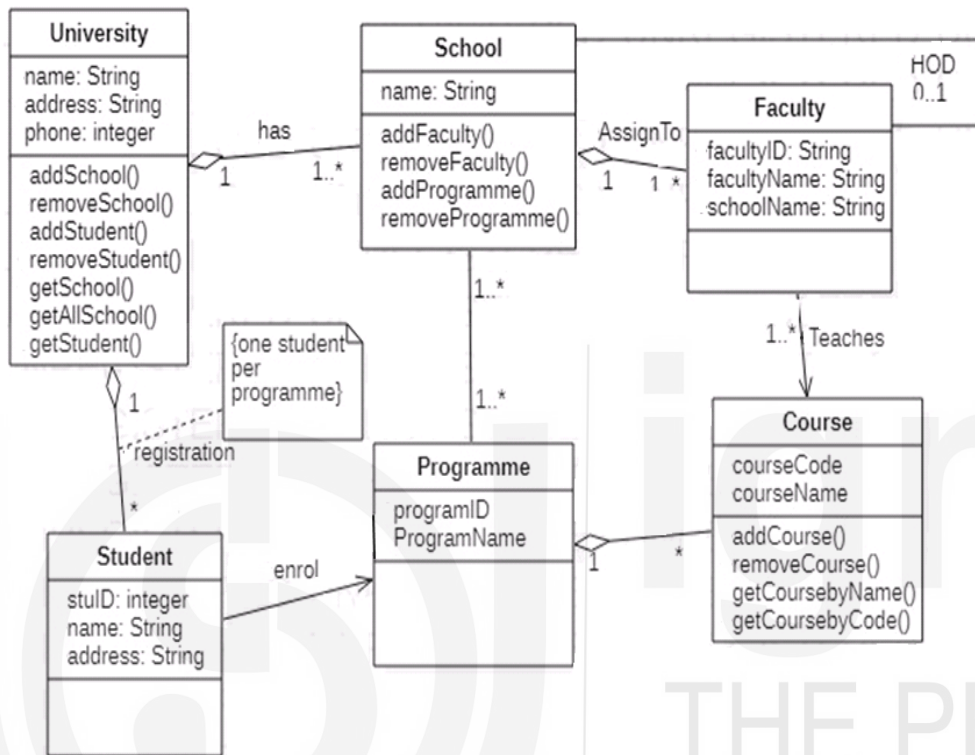


Figure 2.8: Class Diagram for Student Registration

The class diagram is illustrated in figure-2.8 for student registration in courses of programme. This diagram consists of six classes - University, School, Programme, Course, Faculty and Student. The university has many schools and each school have many programmes. Each programme consists of many courses. Faculties are assigned to schools, and these faculties teach many courses. Atleast one faculty works as head of school. One condition for student registration is that one student per programme can only be registered in university. The University and School classes have one to many relationships while School and Programme classes have many to many relationships. Both School and Programme classes stats that many schools contain many programmes. The relationships between classes are shown in figure-2.8.

Check Your Progress 1

1. What is UML? What is the advantage of using UML? Can you name the types of modeling?

2. What are the structural diagrams? Define class diagram.

3. How many types of relationships exist in UML diagrams? Explain each of them with the diagram.

2.4 ADVANCED STRUCTURAL MODELING

In the previous section, you have learnt about the basic elements of structural modeling such as classes, relationships (dependency, association, aggregation and generalization), common mechanisms and the creation of class diagrams. This section will explain some more concepts about structural modeling that are an extension of the previous section.

A class diagram commonly contains classes, interfaces and relationships such as dependency, generalization and association. Class diagram gives static design view of the system. The class diagram and object diagram are examples of the structural model. Both are the same at some point but may have some differences also. Class diagram is a graph of classifier elements connected by their various static relationships, while Object diagram is a graph of instances containing objects and data values. The object diagrams are used to display data structures. A class diagram demonstrate what the objects contain in your system in the form of members and their capacity in terms of methods. In contrast, an object diagram shows how objects will affect each other in your system at some state of the object and what values those objects contain in this state.

Structural model represents a view of a system that focuses on the structure of the objects, including their classifiers, relationships, attributes and operations. The advanced Structural Modeling describes some advanced features of classes and their relationships. Advanced Classes feature contains classifier, visibility and scope. The classifier is a modeling element that defines attributes and operations. It can be accessed by other classifiers through the visibility feature. The following section describes you some more features of the classes as well as advanced relations.

2.5 ADVANCED CLASSES

In the upper section of this Unit, you have learned about classes and drawn a class diagram; it means you are well aware of the classes. This section defines some more

advanced features related to the classes like visibility, scope and multiplicity. Besides these, you will also learn one more term known as classifier.

Classifier

A classifier is a kind of UML element that is used to define some common features in terms of structural (attributes) as well as behavioral (operations). Class, interface, data types, use case, collaboration, node, signal and subsystems all are examples of classifier. A **class** is an important type of **classifier** in UML, which describes a group of objects that share the same features. Some more classifiers are given as follows:

- **Interface:** It is collection of named operations that describe the behavior of an element.
- **Datatype:** its values have no identity, including primitive built-in type (eg. number, string) and enumeration types (eg. Boolean).
- **Node:** It is a run-time physical object that denotes a computational resource such as a server, router etc.
- **Signal:** An asynchronous event communicated between objects.
- **Use case:** It is a list of stages that explain how a process will be carried out in a system.
- **Component:** A component is used to split the actual system into various modular parts.

The graphical illustration of the classifiers is as follows

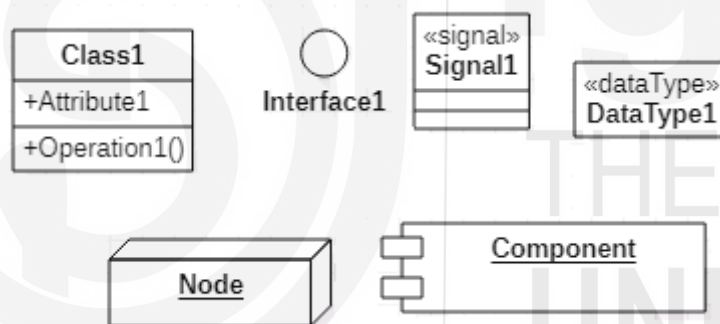


Figure 2.9: Graphical notations of the classifiers

Visibility

By using the visibility feature, a classifier can be accessed by the other classifiers. You can define the visibility of classifier's attributes and operations by using plus sign (+) for public (visible to all elements), negative sign (-) for private (only the classifier itself), hash sign (#) for protected (only by the descendants and classifier itself) and tilde (~) for package (all classifier with the same package). The default visibility of a classifier is public in UML.

Multiplicity

Multiplicity pronounces how many instances of one class may communicate to a single instance of another class. You can display a multiplicity of exactly one (1), zero or one (0..1), many (0..*), or one or more (1..*). You can see this feature in class diagram figure-8.

Scope

The visibility is one of the important features and we have seen it has been applied to attributes and operations of the classifier. In the same direction, another feature also applied to the classifier's attributes and operations is scope. This feature signifies that an attribute or an operation has their existence in all the instances of the classifier or only one copy is available and is shared across all the instances of the classifier. There are two types of scope specifiers as "instances" and "class scope". The Class scope feature is written as underlined on a class diagram. For example it is shown in figure-2.10, the school attribute has default value 'SOCIS' and it is static in nature.

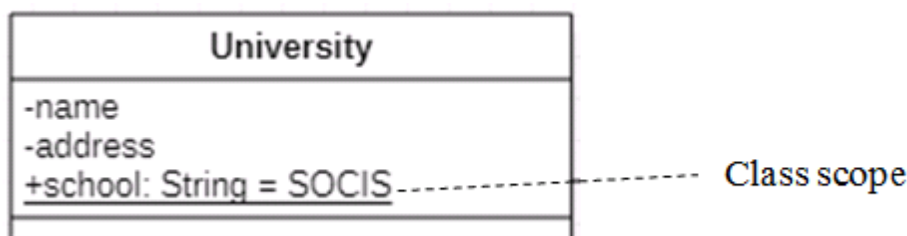


Figure 2.10: Example for class scope

Attributes

When you design classes in UML, you can specify more features like multiplicity, visibility and scope, and the attribute name. Besides these features, you can also specify the type, initial value and variability of each attribute. The attribute specification in UML is as follows:

[visibility] name [multiplicity] [: type] [=initial value] [{property string}]

For example, you can specify the attribute in the following ways:

name – name only
 +name – name and visibility
 name[0..1]:String – name, multiplicity and type

Operations

Like the attribute specification in designed classes, you may also specify features with operations like visibility and scope. Apart from these features, you can also specify the return type, parameter and concurrency. The concurrency feature may have options as guarded, sequential and concurrent. The operation name, parameters list, and return type are collectively called the operation's signature. The operation specification in UML is as follows:

[visibility] name [(parameters-list)] [: return-type] [{property-string}]

For example,

login() – name only
 +login() – name with visibility
 +login() {query} - name and property
 getID: interger – name and return type

You are already familiar with the basic concept of relationships. In the UML diagram, the elements are connected with each other through relationships. So, relationships are the connections between elements. There are three important relationships such as dependency, generalization and association. These three relationships are already explained in the section 2.3.2 of this Unit. This section describes you some more features to these relationships.

Dependency Relationship

In UML modeling, dependency describes the relationships among elements in which one element is dependent on another element. It means that if changes occur in one element, it will affect the other element. This type of relationship comes with named elements such as classes, interfaces, components, packages etc. Dependency is graphically shown as a dashed line.

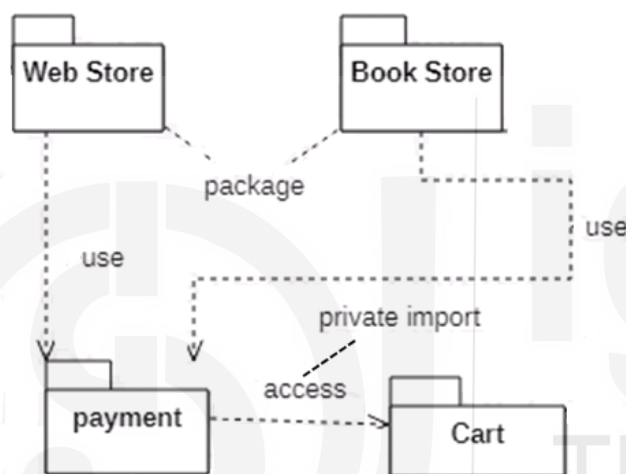


Figure 2.11: Dependency Relationships in Packages.

The Web Store and Book Store package run-through the (depends on) Payment package, as shown in figure-2.11. A **usage** is a dependency relationship in which one element requires another element or set of elements for its operation.

Likewise usage, UML defines a number of keywords or stereotypes that may be used for dependency relationships. The **import** and **access** stereotypes are applied to dependency relationships among packages, as shown in figure-2.16. By using 'access' stereotype, the source package permits access to an element of the target package. The three stereotypes such as **call**, **copy** and **become** are used when modeling interactions among objects. In a similar manner, some stereotypes such as **instanceOf**, **instantiate**, **bind** and **friend** are used to dependency relationships among classes and objects in class diagrams.

Generalization Relationship

In UML diagram, a generalization relationship is a relationship in which one element (the child) is based on another element (the parent). Generalization relationships are used in class, component, deployment and use-case diagrams. This relationship represents that the child class can use the properties and operations of the superclass. The generalization relationships do not have names and can be depicted as a solid line with a hollow arrow pointing to the superclass as shown in figure-2.5.

Association Relationship

An association is a structural relationship; it is used to specify that objects of one element are connected to objects of another element. There are four basic adornments that apply to an association such as name, the role at each end of the association, the multiplicity at each end of the association and aggregation as shown in figure-6.

Aggregation is another form of association, which specifies a whole-part relationship between the aggregate (whole) and the component part. For example, University has many schools. It is an example of aggregation. You can see in figure-2.8 for association and aggregation relationships between classes.

For advanced uses, there are a number of other properties you can use to model refined details such as Navigation, Qualification. A qualified association communicates two object classes with a qualifier. The qualifier is a special attribute that usually decreases the actual multiplicity of an association from many to one but not always. For example, IGNOU University has many regional offices, and there are many officers working. So, this example represents qualified many-to-many associations as shown in figure-2.12.

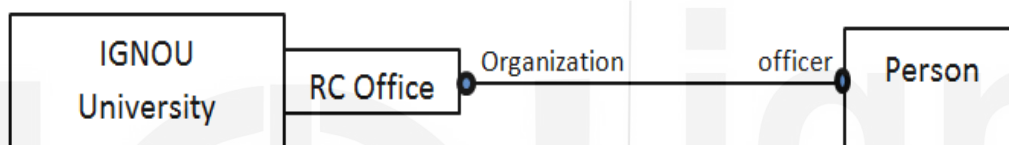


Figure 2.12: Many-to-many Qualification

👉 Check Your Progress 2

1. What are class diagrams in UML? Define the features used for making the class diagrams.

2. Differentiate between generalization and association.

3. What is a qualified association? Explain with an example.

So far, we have seen the classes which display the abstraction. Almost every created class shows abstraction, it means that the class visualizes only those properties which are important and ignores the rest. At times you will need to isolate the implementation of a class from its specification then this can be stated in the UML by using **interfaces**.

You can define any number of operations in **interface** to provide services to a class or component, but you can neither define any attributes nor any implementation of the operations in the **interface**. You must define interface name in such a way that it differentiates it from other interfaces.

A **type** is a stereotype of class used to specify attributes, operations but does not have methods. It is generally used in the analysis process to identify the objects for a system. It is just like conceptual classes because it gives an idea of possible classes. They do not have methods and instances. Some conceptual classes are shown in figure-2.13.



Figure 2.13: Type

A **role** is the behavior of an object contributing to a particular context. In other words, you can say that role is a face that denotes an abstraction that shows to the real world. For example, consider an instance of the class 'Employee'. An employee in a bank may play the role of branch manager, worker and officer, etc. When it deals with a customer, it behaves depending upon the role. When it plays the role of branch manager would represent different properties than if they play as a worker. It is diagrammatically shown as in figure-2.14.

You may define an interface as a stereotyped class by writing its operations, as shown in figure-2.14. You can specify operations with their name, or operation may be augmented to display their full signature and other properties.

Like a class, an interface may play a part in dependency, generalization and association relationships. Besides, an interface may also involve in realization relationships. Realization is a semantic relationship between two classifiers in which one classifier specifies a contract that another classifier guarantees to carry out. Realization is a special kind of relationship, and it differs from others, such as dependency, generalization and association.

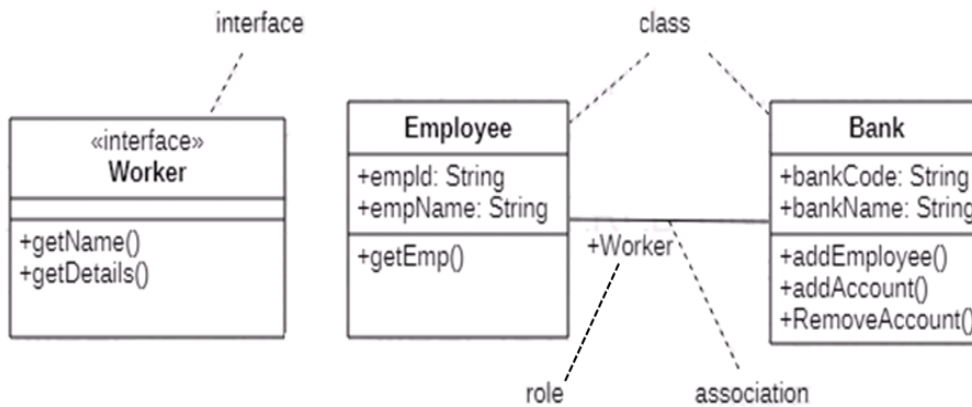


Figure 2.14: Interface and Role

2.7.1 Package Diagram

In the section 2.3.4, you have learnt about the class diagram. Generally, a single class diagram can contain too many classes. UML permits us to place a set of related classes together into a group is called package. It is a kind of structural diagram that displays the system's structure at the level of packages. A package is used for organizing elements into groups. A package can have a **sub package** also. The package diagram can display structure as well as dependencies between sub-systems.

This diagram is used two types of dependencies as import and access. The dependencies are shown by using dotted arrows like in figure-2.15. In **import dependency**, functionality has been imported from one package to another, while **access dependency** designates that one package needs assistance from the functions of another package. For example, many packages have only library files i.e. lib directory and these types of packages are used by other packages. You can just import and use it in your application. It is a depending package that your package needs in order to work. Sometimes your package is accessed by another package, or a package that is dependent on your package is called a **dependent package**. A **plus sign (+)** within a circle is drawn at the end attached to the package indicating that the package has branches or sub-packages.

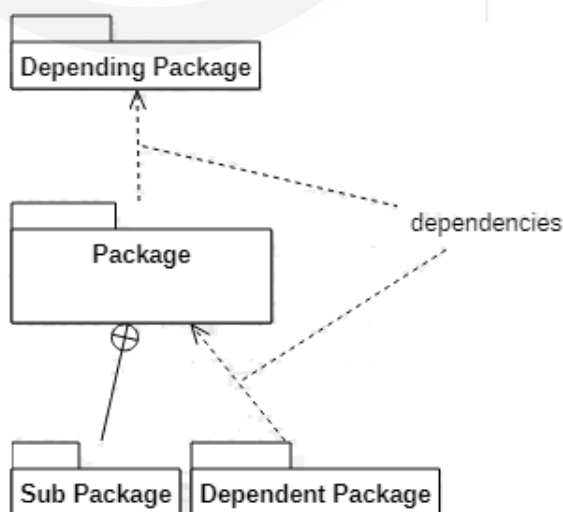


Figure 2.15: Package and dependencies

Package is more helpful to organize a large model. Packages are supported by Java. Classes in one package cannot access the classes and services in another package unless the package is imported.

A package is depicted as a folder and is displayed as a rectangle with a tab at the top left. You must define a package name that differentiates it from other packages; it comes on the tab or inside the rectangle. You can define a package name as a simple text string is known as a simple name or defined as a qualified name. A package may own other elements, including classes, interfaces, components, nodes, collaborations, use cases, diagrams, and other packages. Ownership is a composite relationship which means that the element is declared in the package. If the package is destroyed, it means all the elements of that package are destroyed. Every element is uniquely owned by exactly one package.

Basic components of a package diagram

Package is a namespace used to combine logically related elements within a system and each element should be a packageable element that consists of components, use cases, and packages themselves. The basic components of the packages are package, import, access, use and merge. These basic elements are shown in figure-2.16. A keyword “import” and “access” is used in package diagram as a dashed arrow that indicates accessing type of package. The keyword ‘import’ is used for a public package import and “access” for a private package import. By default, it is public. A package merge component is directed relationship in which the contents of one package are extended by the contents of another package. Basically, the contents of the two packages are logically combined to produce a new package. For example, Cart is a combined package, as shown in figure-2.16.

Visibility of Package

The visibility of the package element can only be public or private, which indicates by "+" for public and "-" for private. Protected or package visibility is not allowed. The public visibility of a package will be visible outside the package, whereas private visibility of a package will not be visible outside.

Package diagram example

The following package diagram illustrates you how packages are interconnected for importing and accessing data. The package diagram shows five elements: package, import, access, use, and merge. The package diagram shown in figure 2.16 consists of Web Store and Book Store packages. These two packages merge the Cart package and use the Payment package. The Payment and Cart package import the packages like the Customer and Inventory package.

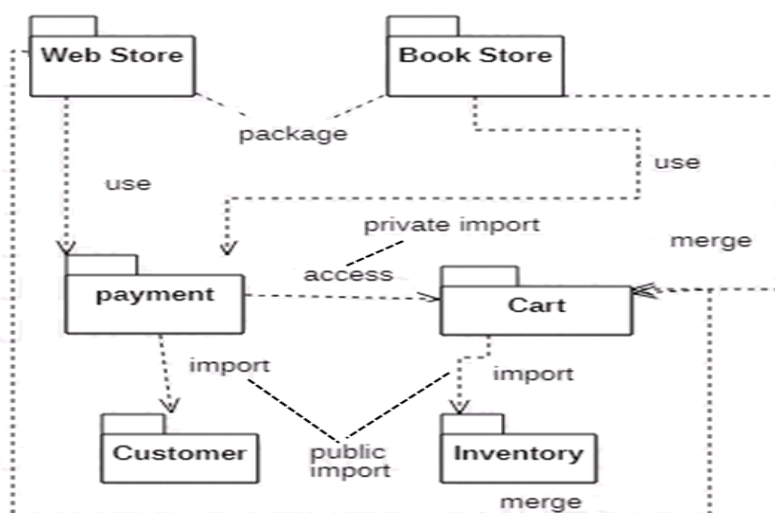


Figure 2.16: Package Diagram

2.7.2 Instance and Object Diagrams

An **instance diagram** defines how a particular group of objects relate to each other. An instance diagram describes object instances. An instance diagram is used to display examples for clarification of complex class diagrams. It is most worthwhile for discussing examples and documenting test cases. A class diagram has an infinite set of instance diagrams. The following figure-2.17 displays a class diagram and its instances. The instance diagram is defined in a rounded box. Objects such as Tata Motors, Hindustan Motors Ltd. and an anonymous 'Car Company' are instances of class Car Company. Similarly, you may think about several instances of Car Company. The class name is written in parenthesis at the top of the box in boldface and object names in normal font.

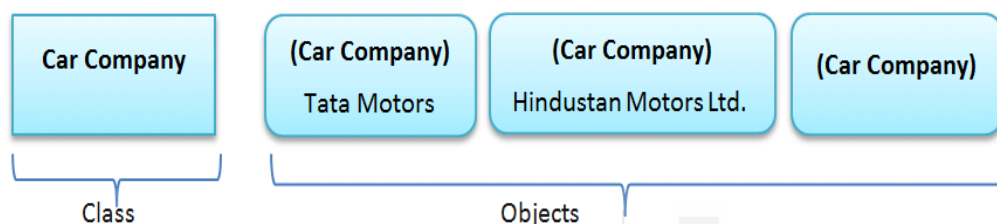


Figure 17: Class and Objects

An **object diagram** defines the instance of a class. It envisions the particular functionality of a system. As you know, objects are real entities, and its behaviour is defined by the classes. You cannot define an object without its class. An object is usually linked to other objects in an object diagram. Notations of object diagram and class diagram are almost similar. The notation of an object is shown in figure-2.18.

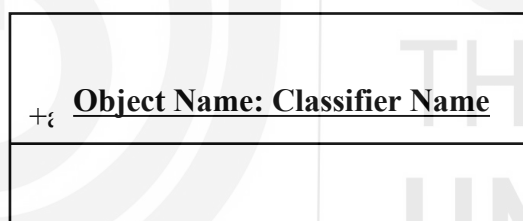
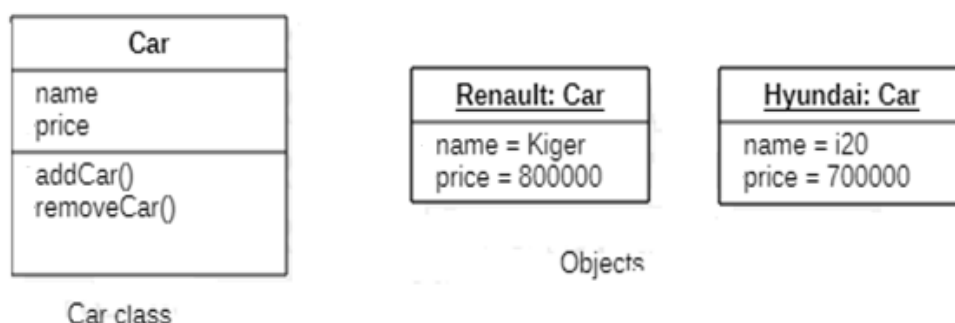


Figure 2.18: Notation for an Object

Object and class diagrams are similar on a certain point, but both have dissimilarity too. The class diagram represents an abstract model containing classes and their relationships, while an object diagram denotes an instance at a particular moment that is concrete. For example, class 'Car' have two attributes as name and price. UML object diagram, as shown in figure-2.19 comprises two objects, namely Renault and Hyundai, which belong to a class Car, similarly many objects belonging to different



manufacturers such as Maruti. Mahindra etc.. may belong to this system as objects.

Figure 2.19: class 'Car' and its corresponding object

Again, it is to reiterate that these objects are instances of the 'Car' class.

OOAD

The following figure-2.20 illustrates you how to prepare an object diagram from the class diagram.

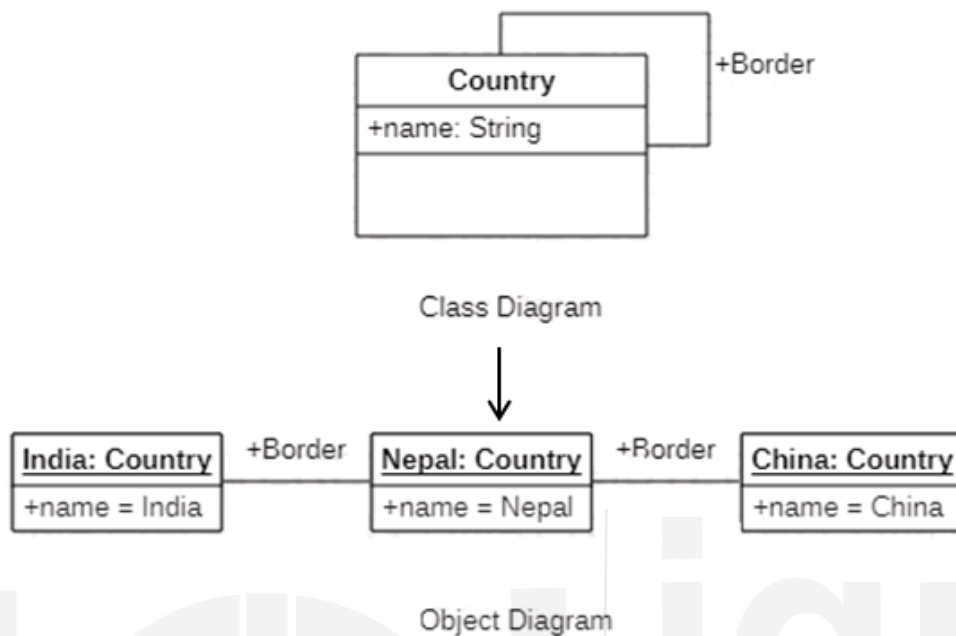


Figure 2.20: Object Diagram

👉 Check Your Progress 3

1. What is modeling? What are the advantages of creating a model? What is the role of interface in UML model?

2. What are package diagrams and their basic components? How many types of dependencies are used in the package diagram?

3. What is Object Diagram? Explain with example.

2.8 SUMMARY

This Unit explained to you the unified modeling language (UML) and their diagrams used in designing a model. Model is a blueprint of the real system that needs to be built. UML is a standard modeling language and is different from the other programming languages. There are many UML development tools like StarUML, ArgoUML, EclipseUML for designing applications. UML diagrams denote two different views of a system model as static and dynamic. The static or structural view focuses on the system's static structure using objects, attributes, operations, and relationships, while the dynamic or behavioural view represents the dynamic nature of the system. This Unit focused on structural modeling, which mainly described classes and their relationships. Relationships are connections between classes. There are three basic types of relationships in UML: Dependencies, Generalisation, and Associations. Some advanced features of classes, as well as relationships, are also discussed in this Unit. Some concepts like interface, type and role are also covered. At the end of the Unit, you are able to design your document with class, object and package diagram.

2.9 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRES

Check Your Progress 1

- 1) Modeling is a verified and well-known technique which helps to build a model. UML (Unified Modeling Language) is a standard modeling language. It is used to build models for software systems. It provides vocabulary and rules used to create a conceptual and physical illustration for any software system.

The advantages of using UML are as follows:

- You may specify the system's structure and behavior, which actually shows the system's blueprint.
- It breaks the large complex system into separate fragments that can be understood easily.
- The new team members can easily grasp it.
- Designed UML model is not a specific platform. So, it is used by any developers who are working on different platforms.

There are three types of modeling in UML: Structural modeling, Behavioral modeling, and Architectural modeling. The Structural modeling describes the static features of a system. The behavioral modeling describes the interaction within the system, while the architectural modeling represents the complete framework of the system; it means that it contains both structural and behavioural elements of the system. An architectural model can be expressed as the blueprint of the entire system.

- 2) The structural diagram gives the static view of the system using objects, attributes, operations and relationships. Structures diagrams do not employ time-connected concepts and do not display the details of dynamic behaviour. There are seven types of structure diagrams: class diagram, object diagram, component diagram, package, deployment, and composite structure diagram.

The Class diagram is the most common diagram used to model object-oriented systems. Class diagram depicts the logical structure of the system. It shows

classes, interfaces and their relationships. When you draw class diagrams, you can use only those class attributes that are important in the current context of your application and display related classes in the same diagrams. It gives the static view of the system. For more details refer to section 2.3.

- 3) Generally, three types of relationships exist in UML diagrams: Dependencies, Generalization, and Association. **Dependencies** are those relationships which occur between two entities when changes occur in the specification of one element may affect another element. **Generalization** is relationships identified in the class-subclass scenario; it is presented when one entity inherits from another. **Association** is “a-part-of” relationships between classes. It describes relationships between classes in which objects of one class is connected to objects of other class for communicating data. Aggregation is a type of association that specifies a “whole-part” relationship between the aggregate (whole) and the component part.

Check Your Progress 2

1. A class diagram is a static structure diagram in the Unified Modeling Language (UML). It defines the structure of a system by displaying the system's classes, their attributes, operations and the relationships (Dependencies, Generalization, and Association) between objects. You can define some advanced features of the classes like visibility, scope, multiplicity.

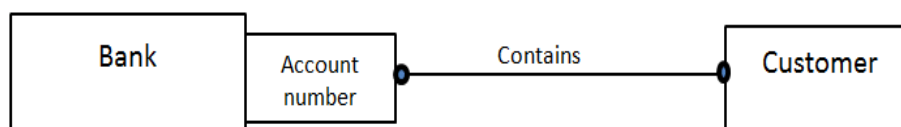
By using visibility feature, a classifier can be accessed by the other classifiers. You can define visibility of classifier's attributes and operations by using plus sign (+) for public (visible to all elements), negative sign (-) for private (only the classifier itself), hash sign (#) for protected (only by the descendants and classifier itself) and tilde (~) for package (all classifier with the same package). The default visibility of a classifier is public in UML.

Multiplicity describes how many instances of one class may communicate to a single instance of other class. You can display a multiplicity of exactly one (1), zero or one (0..1), many (0..*), or one or more (1..*). The scope feature signifies that an attribute or an operation has their existence in all the instances of the classifier or only one copy is available and is shared across all the instances of the classifier. There are two types of scope specifiers such as “instances” and “class scope”. The class scope feature is written as underlined on a class diagram.

2. Generalization is “a-kind-of” relationship. It defines a base or parent class which have some properties functions. A new class will be derived from this base class and it is called child or subclass. The subclass will have access to all the properties and functions of the base or parent class. For example: In a vehicle-car class relationship, vehicle class is parent class while car class is child class.

Association is a special kind of relationship and it is called a "has-a" relationship. It describes relationships between classes in which objects of one class are connected to other class objects for communicating data. When you connect just two classes is called a binary association. Associations can also have more than two classes known as n-ary associations. Each association can have a name, role, multiplicity and aggregation. The special form of association, called aggregation, specifies a “whole-part” relationship between the aggregate (whole) and component part. For example, a paragraph is part of a word document.

3. A qualified association is a UML concept equivalent to a programming concept called associative arrays, maps, and dictionaries. A qualified association has a qualifier that communicates two object classes with a qualifier key. The qualifier is a special attribute that decreases the multiplicity at the target end of the association. For example, Bank has many customers. They may be distinguished in a bank by their account number. The qualified association is shown as in the following figure.



Check Yo **Figure 2.21: Association – Bank and Customer**

1. Modeling is a tested and well-known technique that is used to build a model. Model is a blueprint of the actual system that needs to be built. The model supports visualising the system, identifying the system's structural and behaviour, and giving assistance to make templates for constructing the system. This helps in preparing the design document of the system. You can define any number of operations in the interface to provide services to a class or component. You can neither define any attributes nor any implementation of the operations in the interface.
2. It is a kind of structural diagram that displays the system's structure at the level of packages. A package is used for organizing elements into groups. This diagram uses two main types of dependencies: import and access. The dependencies are shown by using dotted arrows like in the figure drawn. In Import dependency, functionality has been imported from one package to another, while the Access dependency designates that one package needs assistance from the functions of another package.
3. An object diagram is a UML structural diagram. It displays the instances of the classifiers in models. Object diagrams display specific instances of those classifiers and the links between those instances at a point in time. The notation of object diagrams is similar to that used in class diagrams. For the figure of object diagram, please refer to section 2.7.2 .

2.10 REFERENES/FURTHER READING

- Grady Booch, James Rumbaugh and Ivar Jacobson, “The Unified Modeling Language User Guide”, 2nd Edition, Addison-Wesley Object Technology Series, 2005.
- Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, Kelli A. Houston, “Object-Oriented Analysis and Design with Applications,” 3rd Edition, Addison-Wesley, 2007.
- James Rumbaugh, Ivar Jacobson, and Grady Booch, “Unified Modeling Language Reference Manual,” 2nd Edition, Addison-Wesley Professional, 2004.
- John W. Satzinger, Robert B. Jackson, and Stephen D. Burd, “Object-oriented analysis and design with the Unified process,” 1st Edition, Cengage Learning India, 2007.

- Brett McLaughlin, Gary Pollice, and Dave West, “Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D,” Shroff Publisher, First edition, 2006.
- <https://www.uml.org/>
- <https://www.uml-diagrams.org/index-examples.html>
- <https://www.uml-diagrams.org/classifier.html>
- <https://www.uml-diagrams.org/dependency.html>
- <https://www.uml-diagrams.org/association.html>
- <https://www.uml-diagrams.org/package-diagrams-overview.html>
- <https://www.uml-diagrams.org/package-diagrams.html>

OOAD



ignou
THE PEOPLE'S
UNIVERSITY