UNIT 5 AN INTRODUCTION TO UNIFIED MODELING LANGUAGE (UML)

Structure **Page Nos.**

5.0 INTRODUCTION

In the previous units, we have discussed the concepts of object-oriented programming and languages supporting OOP. One of the major assets of object-oriented programming is its reusability. The reusability requires proper design of classes and inheritance hierarchy. To deal with business problems, one needs to do proper design to ensure quick reusable and reliable solution. Object-oriented modeling is an approach to analyze system behaviours, thus, proposing a realistic and proper solution or design to a problem. There are many modeling techniques in this regard. However, a detailed discussion on all such techniques is beyond the scope of this unit. For the purpose of an introduction to object-oriented analysis and design, we have selected unified modeling language (UML) for software development. This unit does not attempt to provide a detailed analysis or design methodology, but is an attempt to make you familiar with some diagrammatic tools that with your knowledge of Software Engineering may be used as a first cut object analysis and design.

5.1 OBJECTIVES

After going though this Unit, you will be able to:

- describe the UML concept;
- present simple design for simple problem, and
- identify various diagrams used in UML.

5.2 WHAT IS UML?

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems.

5.2.1 Goals of UML

The primary goals in the design of the UML are:

1) Provide users with a ready-to-use expressive visual modeling language so that they can develop and exchange meaningful models.
2) Provide extensibility and specialisation mechanisms to extend the core concepts.
3) Be independent of particular programming languages and development processes.
4) Provide a formal basis for understanding the modeling language.
5) Encourage the growth of the OO tools market.
6) Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7) Integrate best practices.

5.2.2 Why Use UML?

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognise the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has increased these architectural problems. The United Modeling Language (UML) was designed to respond to these needs.

5.2.3 History of UML

The development of UML began in late 1994 when Grady Booch and Jim Rumbaugh of Rational Software Corporation began their work on unifying the Booch and OMT (Object Modeling Technique) methods. In the Fall of 1995, Ivar Jacobson and his Objectory company joined rational and this unification effort, merging in the OOSE (Object-Oriented Software Engineering) method.

As the primary authors of the Booch (Grady Booch's work has involved the use of application of data abstraction and information hiding with an emphasis on iterative software development), OMT, and OOSE (Object-Oriented Software Engineering) methods, Grady Booch, Jim Rumbaugh and Ivar Jacobson were motivated to create a unified modeling language for three reasons.

First, these methods were already evolving toward each other independently. It made sense to continue that evolution together rather than apart, that would further confuse users.

Second, by unifying the semantics and notation, they could bring some stability to the object-oriented marketplace, allowing projects to settle on one mature modeling language and letting tool builders focus on delivering more useful features.

Third, they expected that their collaboration would yield improvements in all three earlier methods, helping them to capture lessons learned and to address problems that none of their methods previously handled well.

The efforts of Booch, Rumbaugh, and Jacobson resulted in the release of the UML 0.9 and 0.91. Several organisations saw UML as strategic to their business such as IBM, Object Time, Platinum Technology, Ptech, Taskon, Reich Technologies, these

companies joined the UML partners to contribute their ideas and together the partners produced the UML 1.1.

What is the benefit of UML for users?

UML is based on OMT, Booch, OOSE and other important modelling languages available. It is fusion of OMT, Booch and OOSE techniques for software development and software architecture. Those who have been trained on these three languages will have little trouble getting to work with UML. It provides the opportunity for new integration between tools, processes, and domains. Also it enables developers to focus on delivering business value and provides them a paradigm to accomplish this.

5.2.4 Why do we need UML at all?

As we have described above UML as "UML is a language for specifying, visualising, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems," the question arises why do we need to carry out such an exercise? To answer this, we need to understand the problem domain, its solution domain and the problem solving approach.

Problem can be termed as Requirements of Organisations of their customers. The solutions of the problem is the desired services of products, as the case may be. To deliver value added solutions (maximum quantity and minimum cost and within in the minimum time), organisations must capture knowledge, communicate and leverage knowledge. By capturing knowledge, we mean acquiring it, by communicating it, we mean share it and by leveraging it, we mean utilising it. The main aspect we need to deal here is "communication of sharing" of knowledge. There is an old adage, which says "a picture is worth thousand words", when we express knowledge by visual tools, we are able to deliver (communicate) its contents in a better manner. That's what UML is all about – expressing captured knowledge.

The problem occurs within business context (domain). The solution must also fit in organisation's IT infrastructure. The problem must be fully understood in terms of business requirements and the information system must be fully understood of how it meets those requirements. As we conceptualise the problem and work towards its solution, we capture knowledge (models), make decisions (architectural views) about how we will address different issues and communicate information (diagrams). UML does it all.

5.3 DEFINITIONS

UML concept as we know is based on object-oriented approach; we need to define certain OO concepts before marching into UML's world.

Objects

Objects are the real world models. They are well defined representational constructs. Objects are the physical and conceptual things we find in the world. When something is called as an object in our world, we associate it with a name, properties, etc. Objects encapsulate structural characteristics known as attributes. They contain their own data and programming.

Classes

Classes are descriptions of objects, they contain objects with similar characteristics. Classes encapsulate behavioural characteristics, called operations.

Links

Links are representational constructs that define how classes are related to each other. Links are objects.

Associations

Associations are representational constructs that describe links. Associations are classes.

5.4 THE UML DIAGRAMS

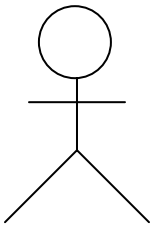
The UML defines various types of Diagrams: Class, Object, Use Case, Sequence, Collaboration, Statechart, Activity, Component and Deployment diagrams.

5.4.1 Use Case Diagrams

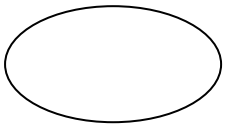
Use Case Diagrams describe the functionality of a system and users of a system. These diagrams contain following items:

Actors – which are users of system, including human beings and other system components.

Use Cases – which includes services provided to users of the system.



Actor



Use Case

An actor represents a user or another system that will interact with the system you are modeling. A use case is an external view of the system that represents some action the user might perform in order to complete a task.

When to Use Cases Diagrams

Use cases are used in almost every project. They are helpful in exposing requirements and planning the project. During the initial stage of a project, most use cases should be defined, but as the project continues more might become visible.

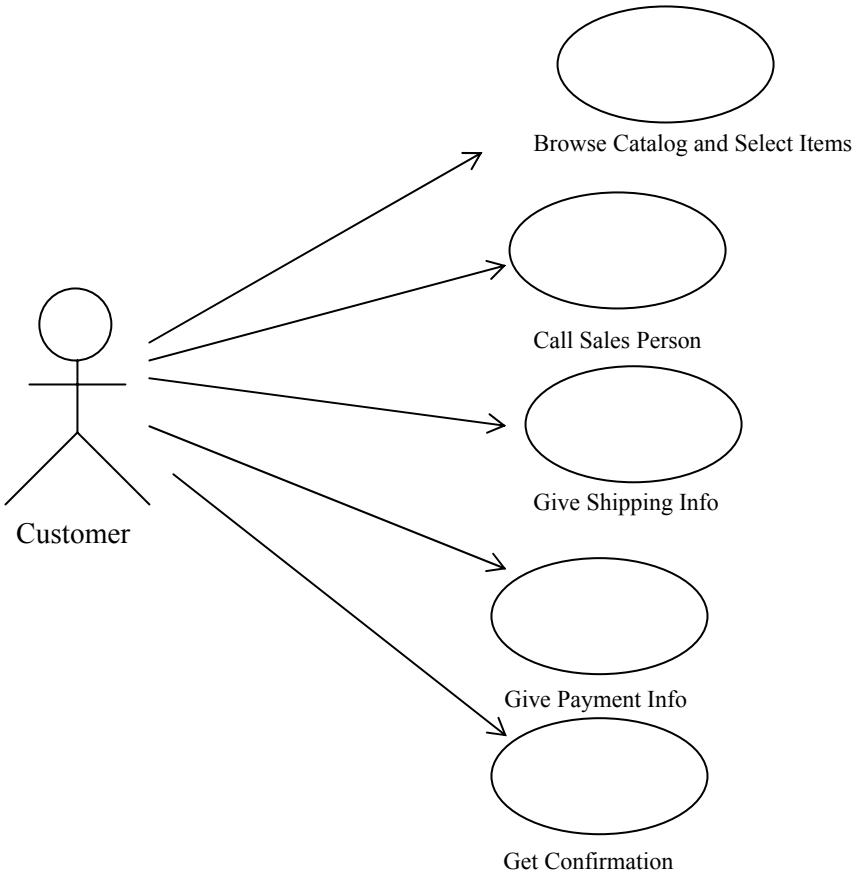
How to Draw Use Cases Diagrams

Use cases are a relatively easy UML diagram to draw.

Start by listing a sequence of steps a user might take in order to complete an action. For example, a user placing an order with a sales company might follow these steps:

1) Browse catalog and select items.
2) Call sales representative.
3) Supply shipping information.
4) Supply payment information.
5) Receive confirmation number from salesperson.

These steps would generate this simple use case diagram:



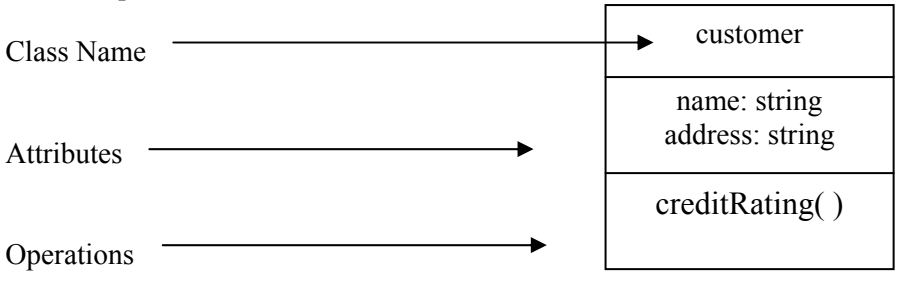
This example shows the customer as an actor because the customer is using the ordering system. The diagram takes the simple steps listed above and shows them as actions the customer might perform.

From this simple diagram, the requirements of the ordering system can easily be derived. The system will need to be able to perform actions for all of the use cases listed. As the project progresses, other use cases might appear. The customer might have a need to add an item to an order that has already been placed. This diagram can easily be expanded until a complete description of the ordering system is derived, capturing all the requirements that the system will need to perform.

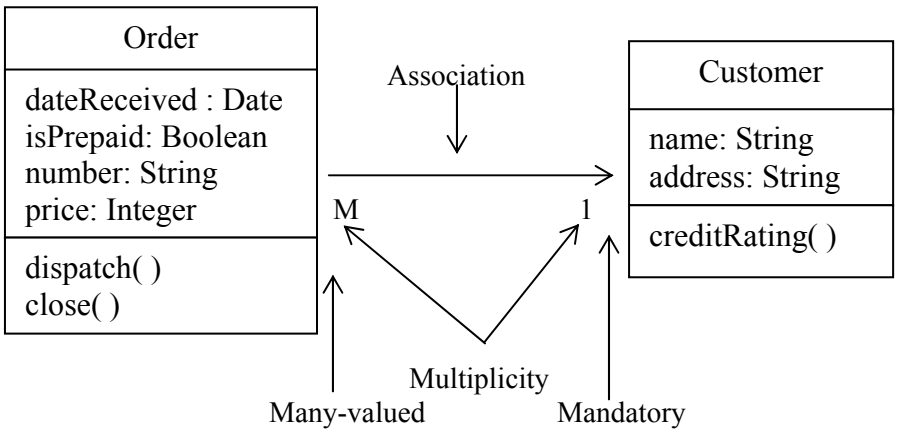
5.4.2 Class Diagrams

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation. These perspectives become evident as the diagram is created and help solidify the design.

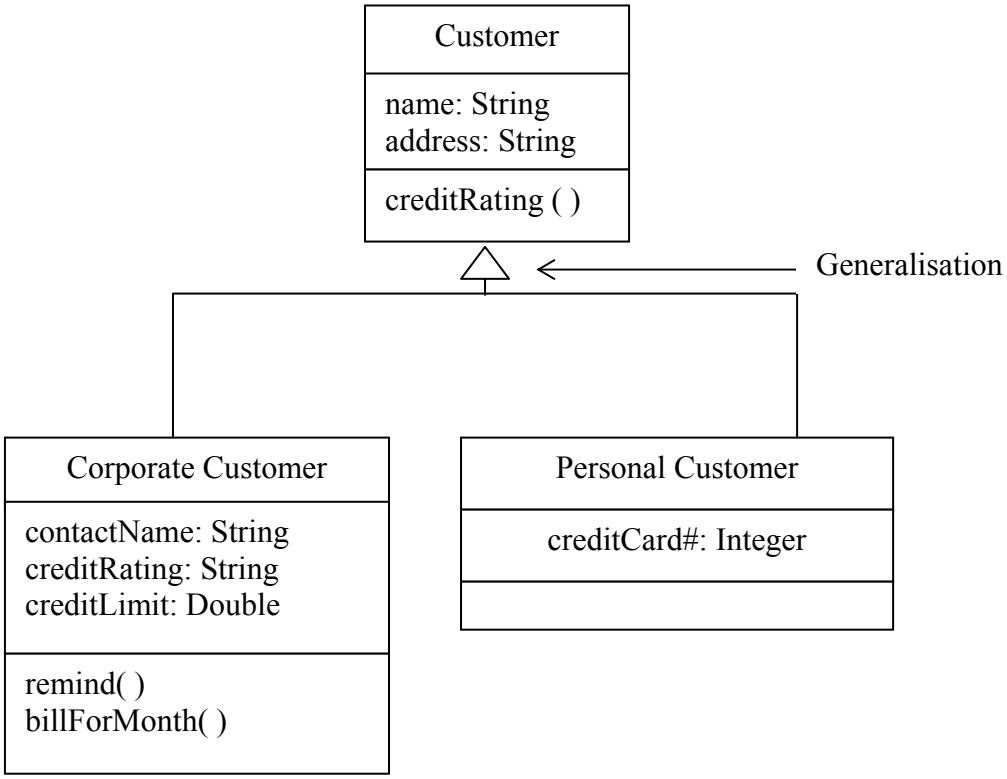
Classes are composed of three things: a name, attributes, and operations/methods. Below is an example of a class:



Class diagrams also display relationships such as containment, inheritance, associations and others. Below is an example of an associative relationship:



The association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes. For example, the class *order* is associated with the class *customer*. The multiplicity of the association denotes the number of objects that can participate in the relationship. For example, an order object can be associated to only one customer, but a customer can be associated to many orders.



Another common relationship in class diagrams is a generalisation. A generalisation is used when two classes are similar, but have some differences. Look at the generalisation below:

In this example, the classes Corporate Customer and Personal Customer have some similarities such as name and address, but each class has some of its own attributes and operations. The class customer is a general form of both the Corporate Customer and Personal Customer classes. This allows the designers to just use one customer class for modules and do not require in-depth representation of each type of customer.

When to Use Class Diagrams

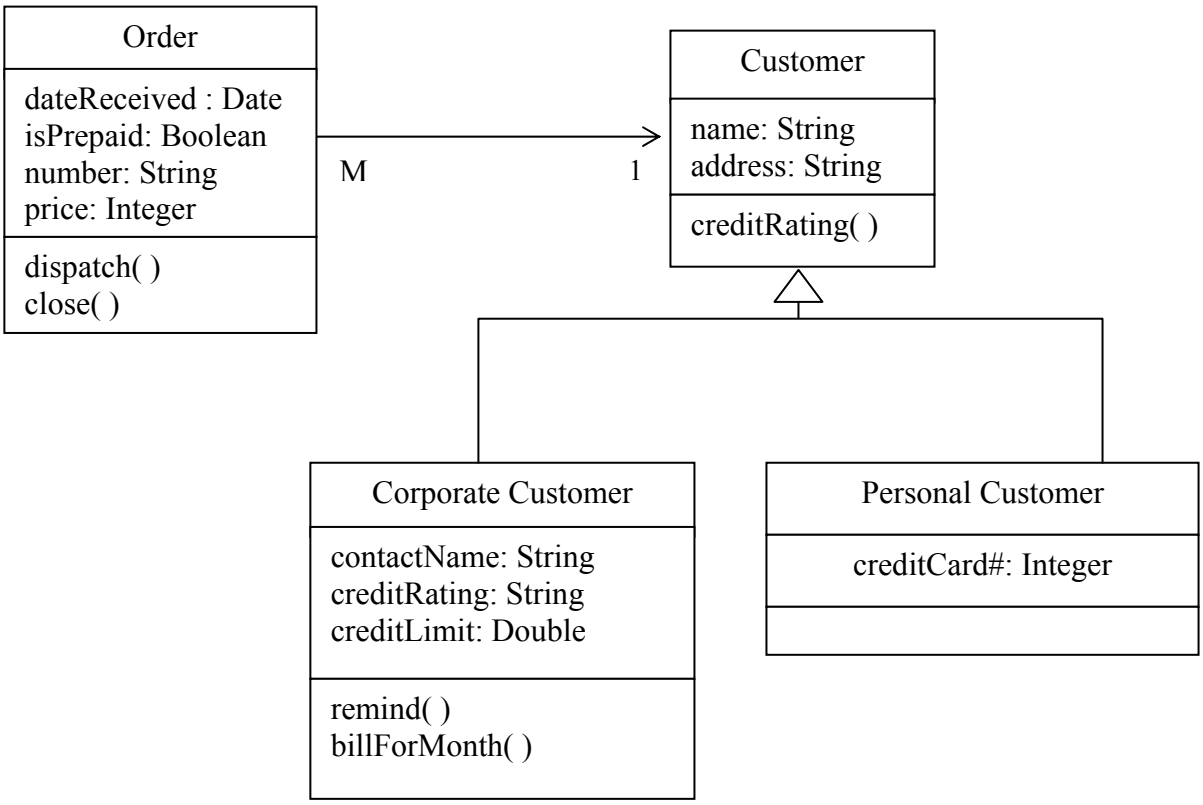
Class diagrams are used in nearly all object-oriented software designs. Use them to describe the classes of the system and their relationships to each other.

How to Draw Class Diagrams

Class diagrams are some of the most difficult UML diagrams to draw. To draw detailed and useful diagrams, a person would have to study UML and object-oriented principles for a long time.

Before drawing a class diagram, consider the three different perspectives of the system the diagram will present; conceptual, specification, and implementation. Try not to focus on one perspective and try to see how they all work together.

When designing classes consider what attributes and operations it will have. Then try to determine how instance of the classes will interact with each other. These are the very first steps of many in developing a class diagram. However, using just these basic techniques, one can develop a complete view of the software system.



5.4.3 Interaction Diagrams

Interaction diagrams model the behaviour of use cases by describing the way groups of objects interact to complete the task. The two kinds of interaction diagrams are sequence and collaboration diagrams.

When to Use: Interaction Diagrams

Interaction diagrams are used when you want to model the behaviour of several objects in a use case. They demonstrate how the objects collaborate for the

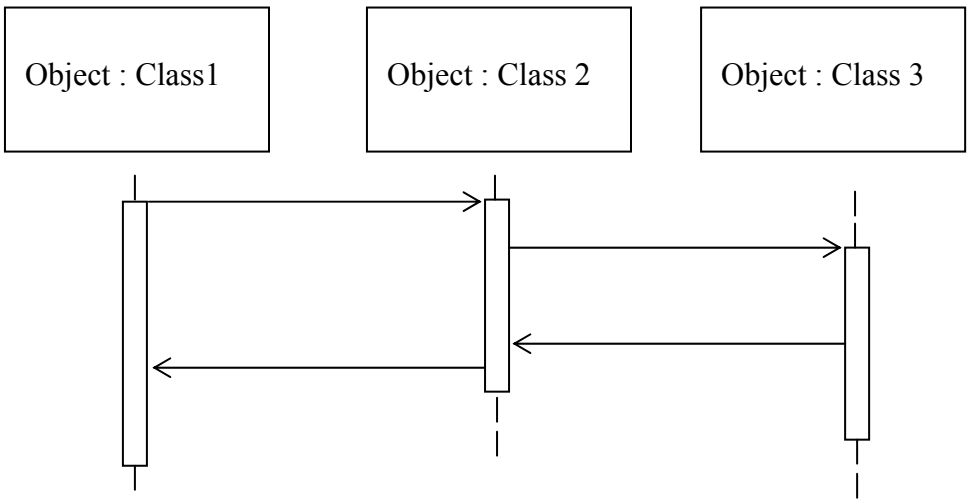
behaviours. Interaction diagrams do not give a in-depth representation of the behaviour. If you want to see what a specific object is doing for several use cases, use a state diagram. To see a particular behaviour over many use cases or threads, use an activity diagrams.

How to Draw Interaction Diagrams

Sequence diagrams, collaboration diagrams, or both diagrams can be used to demonstrate the interaction of objects in a use case. Sequence diagrams generally show the sequence of events that occur. Collaboration diagrams demonstrate how objects are statically connected. Both diagrams are relatively simple to draw and contain similar elements.

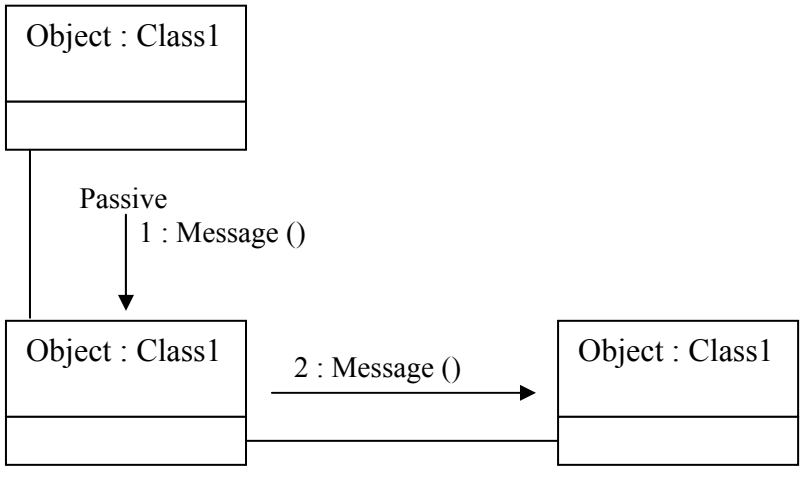
Sequence diagrams

Sequence diagrams describe interactions among classes. These interactions are exchange of messages. Sequence diagrams demonstrate the behaviour of objects in a use case by describing the objects and the messages they pass. The example below shows an object of class 1 start the behaviour by sending a message to an object of class 2. Messages pass between the different objects until the object of class 1 receives the final message.



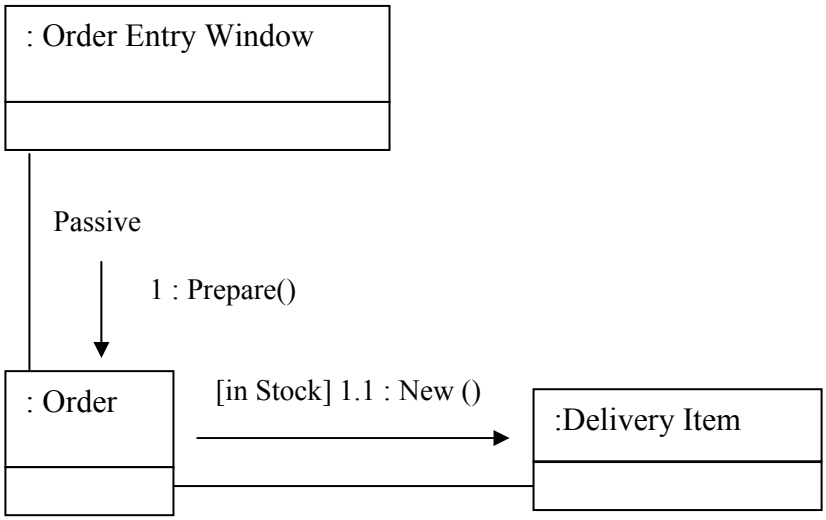
Collaboration diagrams

Collaborations diagrams describe interactions among classes and associations, Collaboration diagrams are also relatively easy to draw. They show the relationship between objects and the order of messages passed between them.



The objects are listed as icons and arrows indicate the messages being passed between them. The numbers next to the messages are called sequence numbers. As the name suggests, they show the sequence of the messages as they are passed between the objects. There are many acceptable sequence numbering schemes in UML. A simple 1,2,3…format can be used, as the example shows, or for more detailed and complex diagrams a 1, 1.1,1.2, 1.2.1… scheme can be used.

The following example shows a simple collaboration diagram for placing an order use case. This time the names of the class appear after the colon, such as Order Entry Window; following the object Name: class Name naming convention, the class name is shown to demonstrate that all of objects of that class will behave the same way.



5.4.4 State Diagrams

State diagrams are used to describe the behaviour of a system, i.e., the states and responses of a class. They describe the behaviour of a class in response to external stimuli. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and track the different states of its objects through the system.

These diagrams contain the following elements:

States which represents the state of an object during its life cycle in which it satisfies some condition.

Transitions which represents relationships between different states of an object.

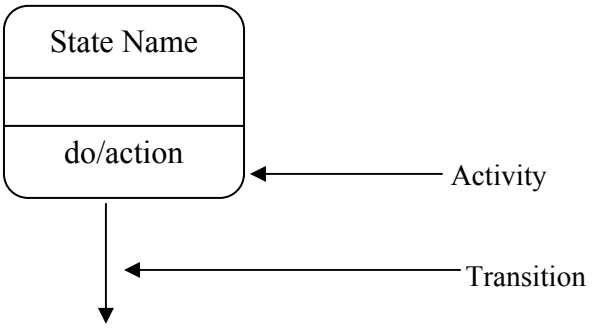
When to Use State Diagrams

State diagrams are used to demonstrate the behaviour of an object through many use cases of the system. Only use state diagrams for classes where it is necessary to understand the behaviour of the object through the entire system. Not all classes will require a state diagram and state diagrams are not useful for describing the collaboration of all objects in a use case. State diagrams are combined with other diagrams such as interaction diagrams and activity diagrams.

How to Draw State Diagrams

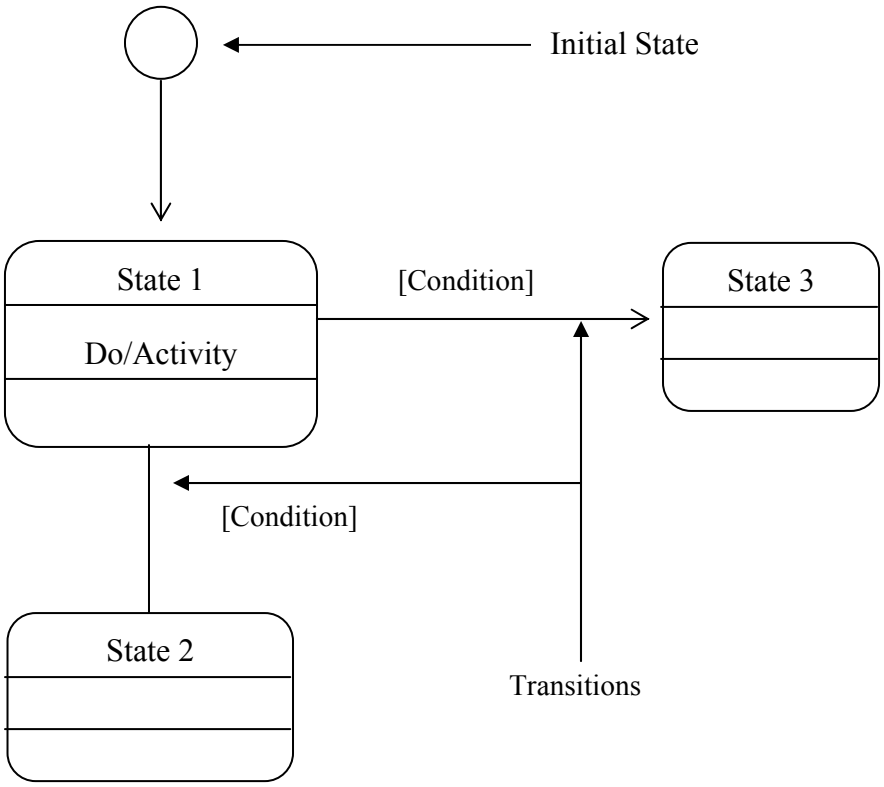
State diagrams have very few elements. The basic elements are rounded boxes representing the state of the object and arrows indicating the transition to the next

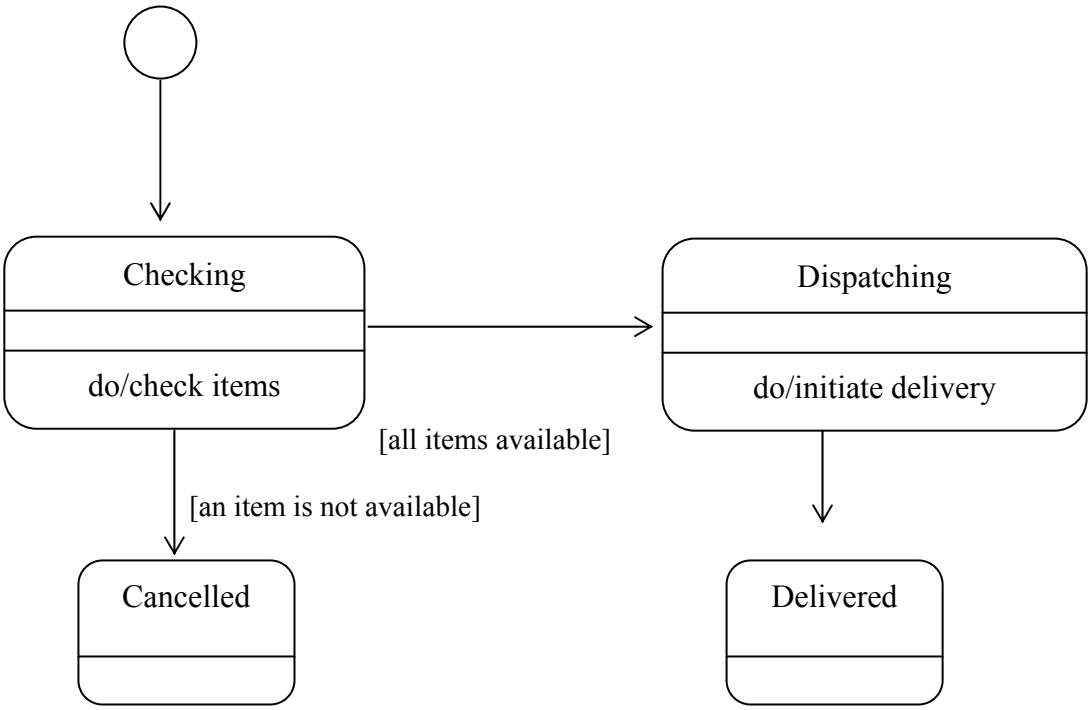
state. The activity section of the state symbol depicts what activities the object will be doing while it is in that state.



All state diagrams being with an initial state of the object. This is the state of the object when it is created. After the initial state, the object begins changing states. Conditions based on the activities can determine what the next state the object transitions to.

Below is an example of a state diagram might look like for an order object. When the object enters the checking state it performs the activity "check items." After the activity is completed, the object transitions to the next state based on the conditions [all items available] or [an item is not available]. If an item is not available, the order is cancelled. If all items are available, then the order is dispatched. When the object transitions to the dispatching state, the activity "initiate delivery" is performed. After this activity is complete, the object transitions again to the delivered state.





5.4.5 Activity Diagrams

Activity diagrams describe the activities of a class. It is similar to state diagram but describes the behaviour of a class in response to internal processing instead of external stimuli. Activity diagrams describe the workflow behaviour of a system. Activity diagrams are similar to state diagram because activities are the state of doing something. The diagrams describe the state of activities by showing the sequence of activities performed. Activity diagrams can show activities that are conditional or parallel.

When to Use Activity Diagrams

Activity diagrams should be used in conjunction with other modeling techniques such as interaction diagram and state diagram. The main reason to use activity diagrams is to model the workflow behind the system being designed. Activity diagrams are also useful for analysing a use case by describing what actions need to take place and when they should occur; describing a complicated sequential algorithm; and modeling applications with parallel processors.

However, activity diagrams should not take the place of interaction diagram and sate diagram. Activity diagrams do not give details about how objects behave or how objects collaborate.

How to Draw Activity Diagrams

Activity diagrams show the flow of activities through the system. Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities. A fork is used when multiple activities are occurring at the same time. The diagram below shows a fork after activity 1. This indicates that both activity 2 and activity 3 are occurring at the same time. After activity 2, there is a branch. The branch describes what activities will take place based on a set of conditions. All branches at some point are followed by a merge to indicate the end of the conditional

behaviour started by that branch. After the merge, all of the parallel activities must be combined by a join before transitioning into the final activity state.

Below is a possible activity diagram for processing an order. The diagram shows the flow of actions in the system's workflow. Once the order is received, the activities split into two parallel sets of activities. One side fills and sends the order while the other handles the billing. On the Fill Order side, the method of delivery is decided conditionally. Depending on the condition either the Overnight Delivery activity or the Regular Delivery activity is performed. Finally, the parallel activities combine to close the order.

5.4.6 Physical Diagrams

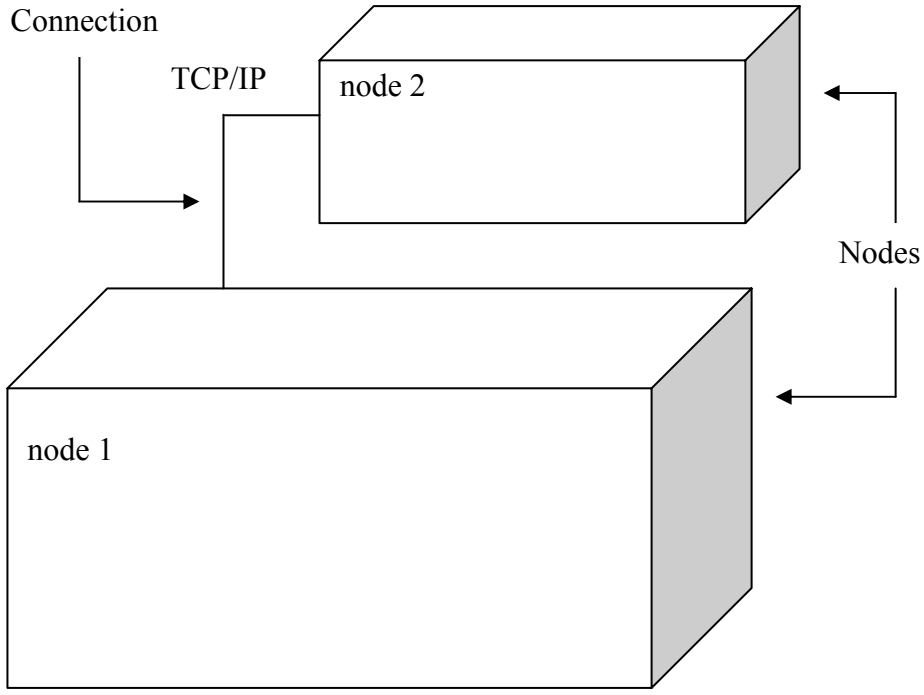
There are two types of physical diagrams: deployment diagrams and component diagrams. Deployment diagrams show the physical relationship between hardware and software in a system. Component diagrams show the software components of a system and how they are related to each other. These relationships are called dependencies.

When to Use Physical Diagrams

Physical diagrams are used when development of the system is complete. Physical diagrams are used to give descriptions of the physical information about a system.

How to Draw Physical Diagrams

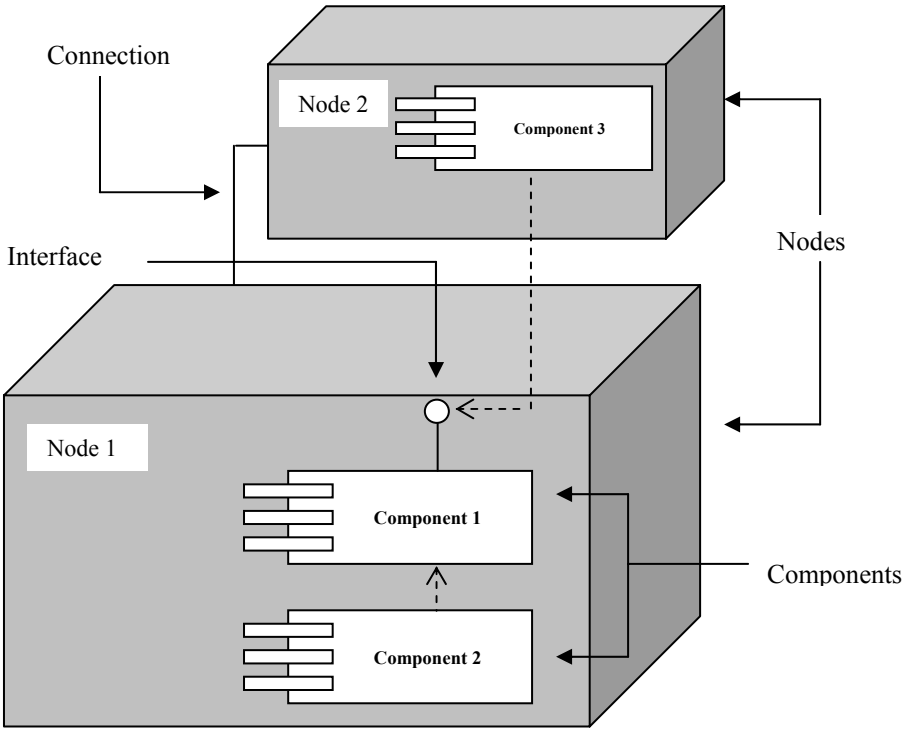
Many times, the deployment and component diagrams are combined into one physical diagram. A combined deployment and component diagram combines the features of both diagrams into one diagram.



The deployment diagram contains nodes and connections. A node usually represents a piece of hardware in the system. A connection depicts the communication path used by the hardware to communicate and usually indicates a method such as TCP/IP.

The component diagram contains components and dependencies. Components represent the physical packaging of a module of code. The dependencies between the components show how changes made to one component may affect the other components in the system. Dependencies in a component diagram are represented by a dashed line between two or more components. Component diagrams can also show the interfaces used by the components to communicate to each other.

The combined deployment and component diagram below gives a high level physical description of the completed system. The diagram shows two nodes which represent two machines communicating through TCP/IP. Component 2 is dependant on component 1, so changes to component 2 could affect component 1. The diagram also depicts component 3 interfacing with component 1. This diagram gives the reader a quick overall view of the entire system.



☞ Check Your Progress

State True T ☐ or False F ☐

1) a) UML is used for waterfall model based implementation. True ☐ False ☐

b) UML increases burden on designers and implements. True ☐ False ☐

c) If you are using UML, then additional documentation tool is not needed. True ☐ False ☐

d) An association in UML is an object. True ☐ False ☐

e) An actor is a user who interacts with the system to be modeled. True ☐ False ☐

f) Class diagrams are very close to E_R diagrams. True ☐ False ☐

g) State diagrams can be used to represent interaction. True ☐ False ☐

h) Activity diagram describes class and objects. True ☐ False ☐

5.5 SUMMARY

In this unit, we have discussed a modeling language, which is getting popular in object-oriented analysis and design. The unit presents the basic objectives of the methodology and various types of diagrams represented in UML. It is advisable that you may explore further readings and web site to look for more examples of UML.

5.6 SOLUTIONS/ANSWERS

Check Your Progress

1) a) False
b) False
c) True
d) False
e) True
f) True
g) False
h) false