RASHTRASANT TUKADOJI MAHARAJ NAGPUR UNIVERSITY, NAGPUR

FOUR YEAR BACHELOR OF TECHNOLOGY (B.Tech.) DEGREE COURSE

SEMESTER: FIFTH (C.B.C.S)

BRANCH: INFORMATION TECHNOLOGY

Subject Name: Software Engineering & Project Management Lab **Subject code: BTIT501P**

Load	Practical	Credits	College Assessment Marks	University Evaluation	Total Marks
2Hrs (Practical)	2	1	25	25	50

Aim: To develop quality software projects for given business system

Prerequisite(s): Basic Programming Knowledge

Course Objectives

1	To understand general idea of software engineering
2	To develop skills to understand creation of object and interaction between them
3	To develop skills required for creation of various UML models

Course Outcome:

At the end of this course students are able to:

CO1	Learn the concept of requirement gathering & to learn the development of use case model
CO2	Understanding the object creation and the interaction between various objects & their collaboration
CO3	Understanding various states of objects & different component views
CO4	Learning the development of various UML models & understanding the complete design phase.

K. D. K. COLLEGE OF ENGINEERING, NAGPUR

IMPARTING TECHNICAL EDUCATION SINCE 1984

NBA Accredited - Five programs:(Civil Engg., Mechanical Engg., Electrical Engg. Comp. Sc. & Engg., Electronics & Telecomm. Engg.) Grade A, NAAC Accredited

---

Department of Information Technology

Semester: v

Name of Subject: Software Engineering & Project Management Lab

Subject Code: BTIT501P

Course Outcome:

At the end of this course students are able to:

CO1	**Learn the concept of requirement gathering & to learn the development of use case model**
CO2	**Understanding the object creation and the interaction between various objects & their collaboration**
CO3	**Understanding various states of objects & different component views**
CO4	**Learning the development of various UML models & understanding the complete design phase.**

LIST OF PRACTICALS

1	To know the Interface (UML).
2	diagram for airline reservation system.
3	To construct the use case diagram for bank management system.
4	To construct the class diagram for hotel management system
5	To construct the Sequence diagram for ATM machine.
6	To construct the collaboration diagram library management System.
7	To construct the activity diagram for access camera surveillance.
8	To construct the state chart diagram for order management system.
9	To construct the component diagram for hotel management system.
10	To construct the deployment for hospital Management System.
11	To construct the entity relationship diagram for student enrolment system.

Dr S.P. Khandait
Subject Teacher

Dr.S.P. Khandait
Head of Department

Experiment No. 1

Aim: To know the Interface (UML).

Description

The heart of object-oriented problem solving is the construction of a model. The model abstracts the essential details of the underlying problem from its usually complicated real world. Several modeling tools are wrapped under the heading of the **UML,** which stands for Unified Modeling Language. The purpose of this course is to present important highlights of the UML.

At the center of the UML are its nine kinds of modeling diagrams, which we describe here.

□ **Use case diagrams**

□ **Class diagrams**

□ **Object diagrams**

□ **Sequence diagrams**

□ **Collaboration diagrams**

□ **State chart diagrams**

□ **Activity diagrams**

□ **Component diagrams**

□ **Deployment diagrams**

The UML is applicable to object-oriented problem solving. Anyone interested in learning UML must be familiar with the underlying tenet of object- oriented problem solving -- it all begins with the construction of a model. A ***model*** is an abstraction of the underlying problem. The ***domain*** is the actual world from which the problem comes. Models consist of ***objects*** that interact by sending each other **messages**. Think of an object as "alive." Objects have things they know (**attributes**) and things they can do (**behaviors** or **operations**). The values of an object's attributes determine its ***state***.

Classes are the "blueprints" for objects. A class wraps attributes (data) and behaviors (methods or functions) into a single distinct entity. Objects are **instances** of classes.

AN INTRODUCTION TO UML DIAGRAM

The Unified Modeling Language is a language for specifying, constructing, visualizing, and documenting the artifacts of a software-intensive system. Analogous to the use of architectural blueprints in the construction industry, UML provides a common language for describing software models, and it can be used in conjunction with a wide range of software lifecycles and development processes.

1. USE CASE DIAGRAM

Use Case diagrams identify the functionality provided by the system (use cases), the users who interact with the system (actors), and the association between the users and the functionality. Use Cases are used in the Analysis phase of software development to articulate the high-level requirements of the system. The primary goals of Use Case diagrams include: □ Providing a high-level view of what the system does

□ Identifying the users ("actors") of the system.

□ Determining areas needing human-computer interfaces.

2. SEQUENCE DIAGRAM

Sequence diagrams document the interactions between classes to achieve a result, such as a use case. Because UML is designed for object-oriented programming, these communications between classes are known as messages. The Sequence diagram lists objects horizontally, and time vertically, and models these messages over time.

NOTATION:

In a Sequence diagram, classes and actors are listed as columns, with vertical lifelines indicating the lifetime of the object over time.

3. ACTIVITY DIAGRAM

Activity diagrams are used to document workflows in a system, from the business level down to the operational level. When looking at an Activity diagram, you'll notice elements from State diagrams. In fact, the Activity diagram is a variation of the state diagram where the "states" represent operations, and the transitions represent the activities that happen when the operation is complete. The general purpose of Activity diagrams is to focus on flows driven by internal processing vs. external events.

4. COMPONENT DIAGRAM

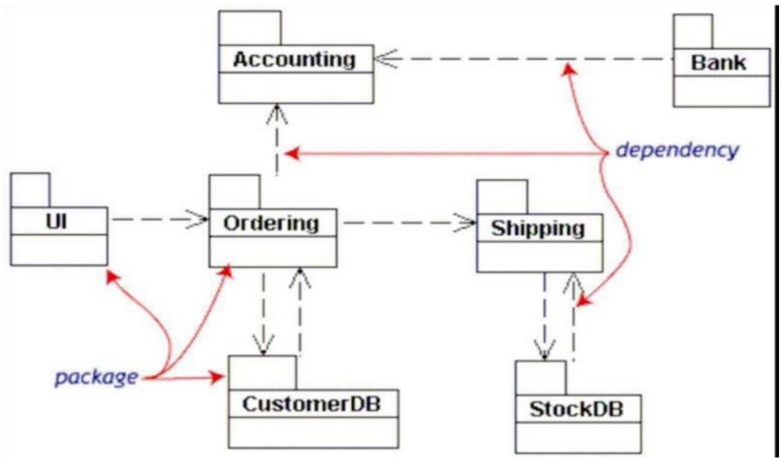
Component diagrams fall under the category of an implementation diagram, a kind of diagram that models the implementation and deployment of the system. A Component Diagram, in particular, is used to describe the dependencies between various software components such as the dependency between executable files and source files. This information is similar to that within make files, which describe source code dependencies and can be used to properly compile an application.

5. CLASS DIAGRAM

A Class diagram gives an overview of a system by showing its classes and the relationships among them. Class diagrams are static -- they display what interacts but not what happens when they do interact.

6. PACKAGES AND OBJECT DIAGRAMS

To simplify complex class diagrams, you can group classes into **packages**. A package is a collection of logically related UML elements. The diagram below is a business model in which the classes are grouped into packages.

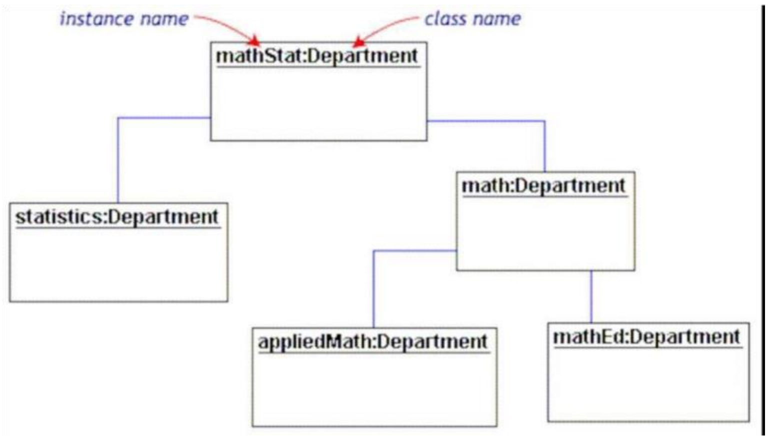


7. OBJECT DIAGRAMS

Object diagrams show instances instead of classes. They are useful for explaining small pieces with complicated relationships, especially recursive relationships.

This small class diagram shows that a university **Department** can contain lots of other **Departments**.

The object diagram below instantiates the class diagram, replacing it by a concrete example.



Each rectangle in the object diagram corresponds to a single instance. Instance names are underlined in UML diagrams. Class or instance names may be omitted from object diagrams as long as the diagram meaning is still clear.

8. COLLABORATION DIAGRAMS

Collaboration diagrams are also interaction diagrams. They convey the same information as sequence diagrams, but they focus on object roles instead of the times that messages are sent. In a sequence diagram, object roles are the vertices and messages are the connecting links. Each message in a collaboration diagram has a **sequence number**. The top- level message is numbered 1. Messages at the same level (sent during the same call) have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur.

9. STATE CHART DIAGRAMS

Objects have behaviours and state. The state of an object depends on its current activity or condition. A **state chart diagram** shows the possible states of the object and the transitions that cause a change in state

10. COMPONENT AND DEPLOYMENT DIAGRAMS

A **component** is a code module. Component diagrams are physical analogs of class diagram. **Deployment diagrams** show the physical configurations of software and hardware. **Result: -**

Hence, we studied the unified modelling language completely and understand it properly.

CONCLUSION:

COURSE OUTCOME	**PROGRAM OUTCOME Attained**
CO1	**PO(1,2,3,5,8,9,10,11,12)**

Experiment No. 2

Aim: To construct the use case diagram for airline reservation system.

DESCRIPTION

To model a system the most important aspect is to capture the dynamic behavior. To clarify a bit in details, *dynamic behavior* means the behavior of the system when it is running /operating. So only static behavior is not sufficient to model a system rather dynamic behavior more important than static behavior. In UML there are five diagrams available to model dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So use case diagrams are consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system. So to model the entire system numbers of use case diagrams are used.

PURPOSE:

The purpose of use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because other four diagrams (activity, sequence, collaboration and State chart) are also having the same purpose. So we will look into some specific purpose which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified. Now when the initial task is complete use case diagrams are modeled to present the outside view.

So in brief, the purposes of use case diagrams can be as follows:

- **Used to gather requirements of a system.**

- **Used to get an outside view of a system.**

- **Identify external and internal factors influencing the system.**

- **Show the interacting among the requirements are actors.**

HOW TO DRAW USE CASE DIAGRAM?

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So we can say that uses cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system. The actors can be human user, some internal applications or may be some external applications. So in a brief when we are planning to draw an use case diagram we should have the following items identified.

- **Functionalities to be represented as an use case**
- **Actors**
- **Relationships among the use cases and actors.**

Use case diagrams are drawn to capture the functional requirements of a system. So after identifying the above items we have to follow the following guidelines to draw an efficient use case diagram.

- **The name of a use case is very important. So the name should be chosen in such a way so that it can identify the functionalities performed.**
- **Give a suitable name for actors.**
- **Show relationships and dependencies clearly in the diagram.**
- **Do not try to include all types of relationships. Because the main purpose of the diagram is to identify requirements.**
- **Use note when ever required to clarify some important points.**

The following is a sample use case diagram representing the order management system. So if we look into the diagram then we will find three use cases (Order, SpecialOrder and Normal Order) and one actor which is customer.

The *SpecialOrder* and *NormalOrder* use cases are extended from *Order* use case. So they have extends relationship. Another important point is to identify the system boundary which is shown in the picture. The actor *Customer* lies outside the system as it is an external user of the system**.**

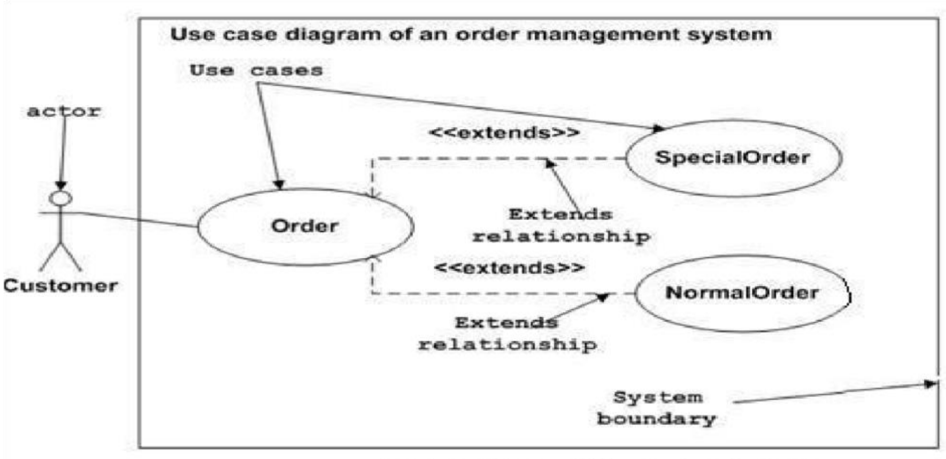


Figure: Sample Use Case diagram

WHERE TO USE CASE DIAGRAMS?

As we have already discussed there are five diagrams in UML to model dynamic view of a system. Now each and every model has some specific purpose to use. Actually these specific purposes are different angles of a running system.

So to understand the dynamics of a system we need to use different types of diagrams. Use case diagram is one of them and its specific purpose is to gather system requirements and actors.

Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output and the function of the black box is known.

These diagrams are used at a very high level of design. Then this high level design is refined again and again to get a complete and practical picture of the system. A well structured use case also describes the precondition, post condition, exceptions. And these extra elements are used to make test cases when performing the testing.

Although the use cases are not a good candidate for forward and reverse engineering but still they are used in a slight different way to make forward and reverse engineering. And the same is true for reverse engineering. Still use case diagram is used differently to make it a candidate for reverse engineering.

In forward engineering use case diagrams are used to make test cases and in reverse engineering use cases are used to prepare the requirement details from the existing application.

So, the following are the places where use case diagrams are used:

- **Requirement analysis and high-level design.**

- **Model the context of a system.**

- **Reverse engineering.**

- **Forward engineering.**

Result:

Hence, we studied and constructed use case diagram.

CONCLUSION: COURSE OUTCOME	PROGRAM OUTCOME Attained
CO1	PO (1,2,3,5,8,9,10,11,12)

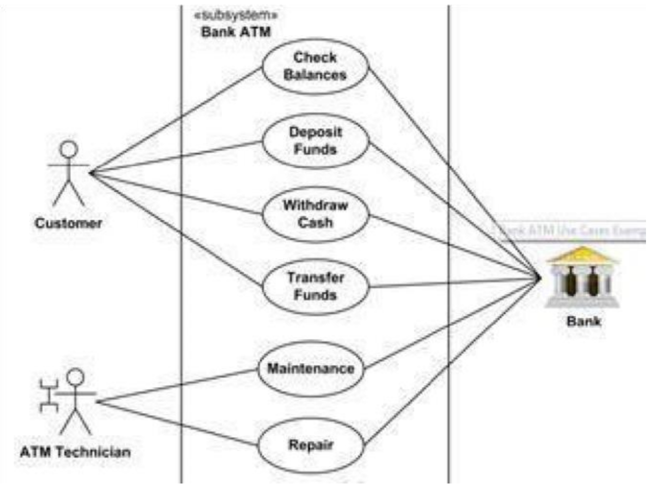
Experiment No. 3

Aim: To construct the use case diagram for bank management system.

DESCRIPTION

To model a system the most important aspect is to capture the dynamic behavior. To clarify a bit in details, *dynamic behavior* means the behavior of the system when it is running /operating. So only static behavior is not sufficient to model a system rather dynamic behavior more important than static behavior. In UML there are five diagrams available to model dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So use case diagrams are consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system. So to model the entire system numbers of use case diagrams are used.



Result:

Hence, we studied and constructed use case diagram.

CONCLUSION:

COURSE OUTCOME	PROGRAM OUTCOME Attained
CO1	PO(1,2,3,5,8,9,10,11,12

Experiment No. 4

Aim: To construct the class diagram for hotel management system.

DESCRIPTION

The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages.The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a *structural diagram*.

PURPOSE:

The purpose of the class diagram is to model the static view of an application. The class diagrams are the only diagrams which can be directly mapped with object oriented languages and thus widely used at the time of construction.

The UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application but class diagram is a bit different. So it is the most popular UML diagram in the coder community.

So the purpose of the class diagram can be summarized as:

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

HOW TO DRAW CLASS DIAGRAM?

Class diagrams are the most popular UML diagrams used for construction of software KDKCE NAGPUR SUB: - SEPM V SEM (IT)applications. So it is very important to learn the drawing procedure of class diagram. Class diagrams have lot of properties to consider while drawing but here the diagram will be considered from a top level view.

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system.

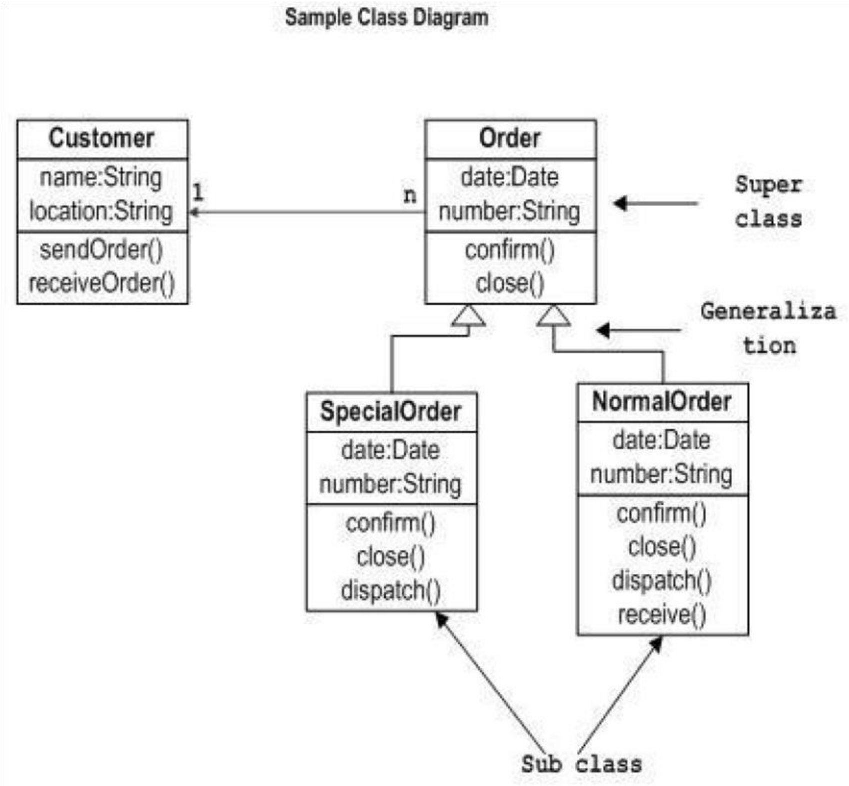
The following points should be remembered while drawing a class diagram:

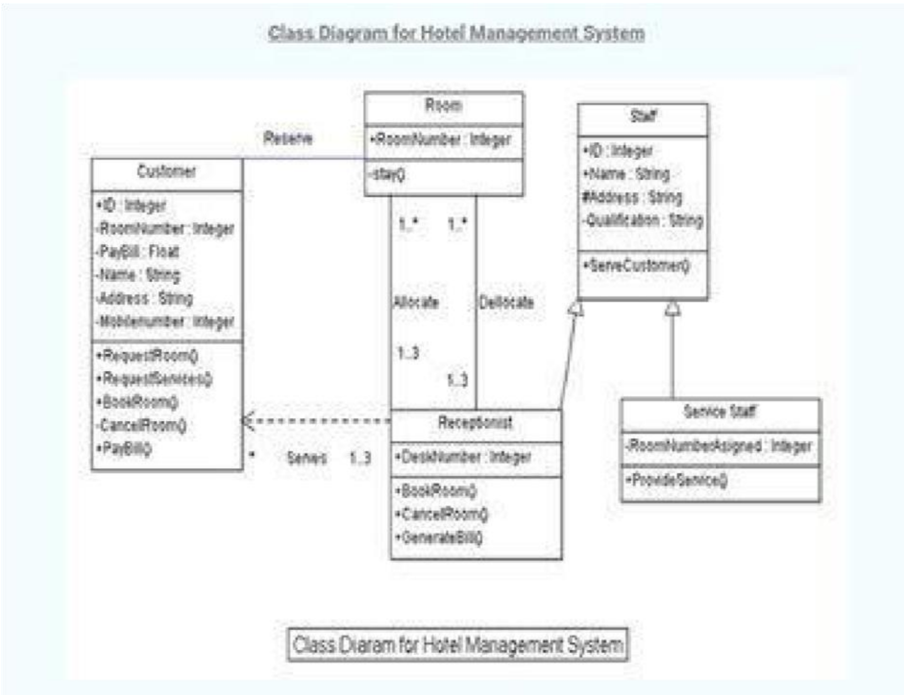
The name of the class diagram should be meaningful to describe the aspect of the system.Each element and their relationships should be identified in advance.Responsibility (attributes and methods) of each class should be clearly identified.For each class minimum number of properties should be specified. Because unnecessary properties will make the diagram complicated.Use notes when ever required to describe some aspect of the diagram. Because at the end of the drawing it should be understandable to the developer/coder.

Now the following diagram is an example of an *Order System* of an application. So it describes a particular aspect of the entire application.

- First of all *Order* and *Customer* are identified as the two elements of the system and they have a *one to many* relationship because a customer can have multiple orders.
- We would keep *Order* class is an abstract class and it has two concrete classes (inheritance relationship) *SpecialOrder* and *NormalOrder*.
- The two inherited classes have all the properties as the *Order* class. In addition they have additional functions like *dispatch ()* and *receive ()*.

So the following class diagram has been drawn considering all the points mentioned above:





Result:Hence we studied and constructed class diagram.

CONCLUSION: COURSE OUTCOME	PROGRAM OUTCOME Attained
CO1	PO(1,2,3,5,8,9,10,11,12)

Experiment No . 5

Aim: To construct the Sequence diagram for ATM machine.

DESCRIPTION

UML sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modeling, which focuses on identifying the behavior within your system. Other dynamic modeling techniques include activity diagramming, communication diagramming, timing diagramming, and interaction overview diagramming. Sequence diagrams, along with class diagrams and physical data models are in my opinion the most important design-level models for modern business application development.

Sequence diagrams are typically used to model:

1. **Usage scenarios**. A usage scenario is a description of a potential way your system is used. The logic of a usage scenario may be part of a use case, perhaps an alternate course. It may also be one entire pass through a use case, such as the logic described by the basic course of action or a portion of the basic course of action, plus one or more alternate scenarios. The logic of a usage scenario may also be a pass through the logic contained in several use cases. For example, a student enrolls in the university, and then immediately enrolls in three seminars.
2. **The logic of methods**. Sequence diagrams can be used to explore the logic of a complex operation, function, or procedure. One way to think of sequence diagrams, particularly highly detailed diagrams, is as visual object code.
3. **The logic of services**. A service is effectively a high-level method, often one that can be invoked by a wide variety of clients. This includes web- services as well as business transactions implemented by a variety of technologies such as CICS/COBOL or CORBAcompliant object request brokers (ORBs).

PURPOSE

The sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur. Much like the class diagram, developers typically think sequence diagrams were meant exclusively for them. However, an organization's business staff can find sequence diagrams useful to communicate how the business currently works by showing how various business objects interact. Besides documenting an organization's current affairs, a business-level sequence diagram can be used as a requirements document to communicate requirements for a future system implementation. During the requirements phase of a project, analysts can take use cases to the next level by providing a more formal level of refinement. When that occurs, use cases are often refined into one or more sequence diagrams.

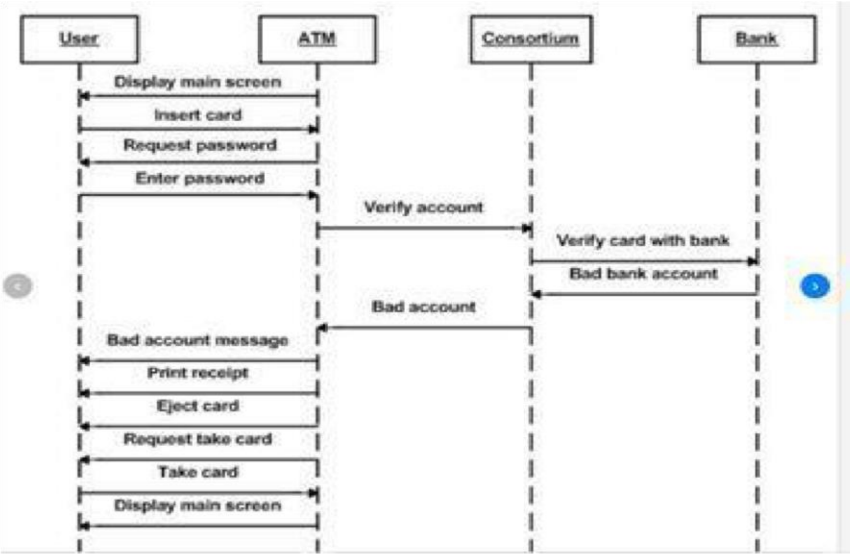
An organization's technical staff can find sequence diagrams useful in documenting how a future system should behave. During the design phase, architects and developers can use the diagram to force out the system's object interactions, thus fleshing out overall system design. One of the primary uses of sequence diagrams is in the transition from requirements expressed as use cases to the next and more formal level of refinement. Use cases are often refined into one or more sequence diagrams. In addition to their use in designing new systems, sequence diagrams can be used to document how objects in an existing (call it "legacy") system currently interact. This documentation is very useful when transitioning a system to another person or organization.

SEQUENCE DIAGRAM COMPONENTS

To understand what a sequence diagram is, you should be familiar with its components.

Sequence diagrams are made up of the following elements:

- **Object** - this box shape represents a class, or object, in UML. They demonstrate how an object will behave in the context of the system. Class attributes should not be listed in this shape.
- **Activation boxes** - symbolized by a rectangle shape, an activation box represents the time needed for an object to complete a task. The longer the task will take, the longer the activation box becomes.
- **Actors** - represented by a stick figure, actors are entities that are both interactive with and external to the system.
- **Package** - also known as a frame, this is a rectangle shape that is used in UML 2.0 notation to contain interactive elements of the diagram. The shape has a small inner rectangle for labeling the diagram.
- **Lifeline** - a dashed vertical line that represents the passage of time as it extends downward. Along with time, they represent the sequential events that occur to an object during the charted process. Lifelines may begin with a labeled rectangle shape or an actor symbol. ● **Option loops** - a rectangle shape with a smaller label within it. This symbol is used to model "if then" scenarios, i.e., a circumstance that will only occur under certain conditions. ● **Alternatives** - used to symbolize a choice (that is usually mutually exclusive) between two or more message sequences. To represent alternatives, use the labeled rectangle shape with a dashed line inside.
- **Messages** - packets of information that are transmitted between objects. They may reflect the start and execution of an operation, or the sending and reception of a signal.



Result: Hence we have created sequence diagram for ATM

CONCLUSION:

COURSE OUTCOME	PROGRAM OUTCOME Attained
CO2	PO(1,2,3,4,5,8,9,10,11,12)

Experiment No. 6

Aim: To construct the collaboration diagram library management System.

DESCRIPTION

It shows the object organization as shown below. Here in collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization where as the collaboration diagram shows the object organization.Now to choose between these two diagrams the main emphasis is given on the type of requirement. If the time sequence is important then sequence diagram is used and if organization is required then collaboration diagram is used.

OBJECTS

An object is represented by an object symbol showing the name of the object and its class underlined, separated by a colon:

Object name : class name

You can use objects in collaboration diagrams in the following ways:

- An object's class can be unspecified. Normally you create a collaboration diagram with objects first and specify their classes later.
- The objects can be unnamed, but you should name them if you want to discriminate different objects of the same class.
- An object's class can itself be represented in a collaboration diagram, if it actively participates in the collaboration.

ACTORS

Normally an actor instance occurs in the collaboration diagram, as the invoker of the interaction. If you have several actor instances in the same diagram, try keeping them in the periphery of the diagram.

LINKS

Links are defined as follows:

- A link is a relationship among objects across which messages can be sent. In collaboration diagrams, a link is shown as a solid line between two objects.
- An object interacts with, or navigates to, other objects through its links to these objects. ● A link can be an instance of an association, or it can be **anonymous**, meaning that its association is unspecified.
- Message flows are attached to links, see **Messages**.

MESSAGES

A message is a communication between objects that conveys information with the expectation that activity will ensue. In collaboration diagrams, a message is shown as a labeled arrow placed near a link. This means that the link is used to transport, or otherwise implement the delivery of the message to the target object. The arrow points along the link in the direction of the target object (the one that receives the message). The arrow is labeled with the name of the message, and its parameters. The arrow may also be labeled with a sequence number to show the sequence of the message in the overall interaction. Sequence numbers are often used in collaboration diagrams, because they are the only way of describing the relative sequencing of messages.

HOW TO CREATE COLLABORATION DIAGRAM?

While creating a collaboration diagram, place the most important objects involved in the collaboration in the middle of the diagram. This helps set the stage for clearly showing the relationships between collaborating objects.

When designing collaboration diagrams from scratch (versus generating them automatically from sequence diagrams), here are the basic steps to follow:

- Determine the scope of the diagram. As with sequence diagrams, the scope of a collaboration diagram can be a use case.
- Place the objects that participate in the collaboration on the diagram. Remember to place the most important objects towards the center of the diagram.
- If a particular object has a property or maintains a state that is important to the collaboration, set the initial value of the property or state.
- Create links between the objects.
- Create messages associated with each link.
- Add sequence numbers to each message corresponding to the time-

COURSE OUTCOME	PROGRAM OUTCOME Attained
CO2	PO(1,2,3,4,5,8,9,10,11,12)

Result: Thus we have created collaboration diagram for library management System
Experiment No. 7

Aim: To construct the activity diagram for access camera surveillance.

DESCRIPTION

Activity diagram is another important diagram in UML to describe dynamic aspects of the system.

Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system.

So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc.

PURPOSE:

The basic purposes of activity diagrams are similar to other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

So the purposes can be described as:

● Draw the activity flow of a system.

● Describe the sequence from one activity to another.

● Describe the parallel, branched and concurrent flow of the system.

HOW TO DRAW ACTIVITY DIAGRAM?

Activity diagrams are mainly used as a flow chart consists of activities performed by the system. But activity diagram are not exactly a flow chart as they have some additional capabilities. These additional capabilities include branching, parallel flow, swimlane etc. Before drawing an activity diagram we must have a clear understanding about the elements used in activity diagram. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities we need to understand how they are associated with constraints and conditions.

So before drawing an activity diagram we should identify the following elements: ● Activities

● Association

● Conditions

● Constraints

Once the above mentioned parameters are identified we need to make a mental layout of the entire flow. This mental layout is then transformed into an activity diagram.

The following is an example of an activity diagram for order management system. In the diagram four activities are identified which are associated with conditions. One important point should be clearly understood that an activity diagram cannot be exactly matched with the code.

The activity diagram is made to understand the flow of activities and mainly used by the business users.

The following diagram is drawn with the four main activities:

● Send order by the customer

● Receipt of the order

● Confirm order

● Dispatch order

After receiving the order request condition checks are performed to check if it is normal or special order. After the type of order is identified dispatch activity is performed and that is marked as the termination of the process.

WHERE TO USE ACTIVITY DIAGRAMS?

The basic usage of activity diagram is similar to other four UML diagrams. The specific usage is to model the control flow from one activity to another. This control flow does not include messages.

The activity diagram is suitable for modeling the activity flow of the system. An application can have multiple systems. Activity diagram also captures these systems and describes flow from one system to another. This specific usage is not available in other diagrams. These systems can be database, external queues or any other system.

Now we will look into the practical applications of the activity diagram. From the above discussion it is clear that an activity diagram is drawn from a very high level. So it gives high level view of a system. This high level view is mainly for business users or any other person who is not a technical person.

This diagram is used to model the activities which are nothing but business requirements. So the diagram has more impact on business understanding rather implementation details.

Result:Hence we studied and constructed activity diagram.

COURSE OUTCOME	PROGRAM OUTCOME Attained
CO2	PO(1,2,3,4,5,8,9,10,11,12)

Experiment No. 8

Aim: To construct the state chart diagram for order management system.

DESCRIPTION

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.

A State chart diagram describes a state machine. Now to clarify it state machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

Activity diagram explained in next chapter is a special kind of a State chart diagram. As State chart diagram defines states it is used to model lifetime of an object.

PURPOSE:

State chart diagram is one of the five UML diagrams used to model dynamic nature of a system. They define different states of an object during its lifetime. And these states are changed by events. So State chart diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. So the most important purpose of State chart diagram is to model life time of an object from creation to termination.

State chart diagrams are also used for forward and reverse engineering of a system. But the main purpose is to model reactive system.

Following are the main purposes of using State chart diagrams: ●
To model dynamic aspect of a system.

● To model life time of a reactive system.

● To describe different states of an object during its life time.

● Define a state machine to model states of an object.

HOW TO DRAW STATE CHART DIAGRAM?

State chart diagram is used to describe the states of different objects in its life cycle. So the emphasis is given on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

State chart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

Before drawing a State chart diagram we must have clarified the following points: ●
Identify important objects to be analyzed.

● Identify the states.

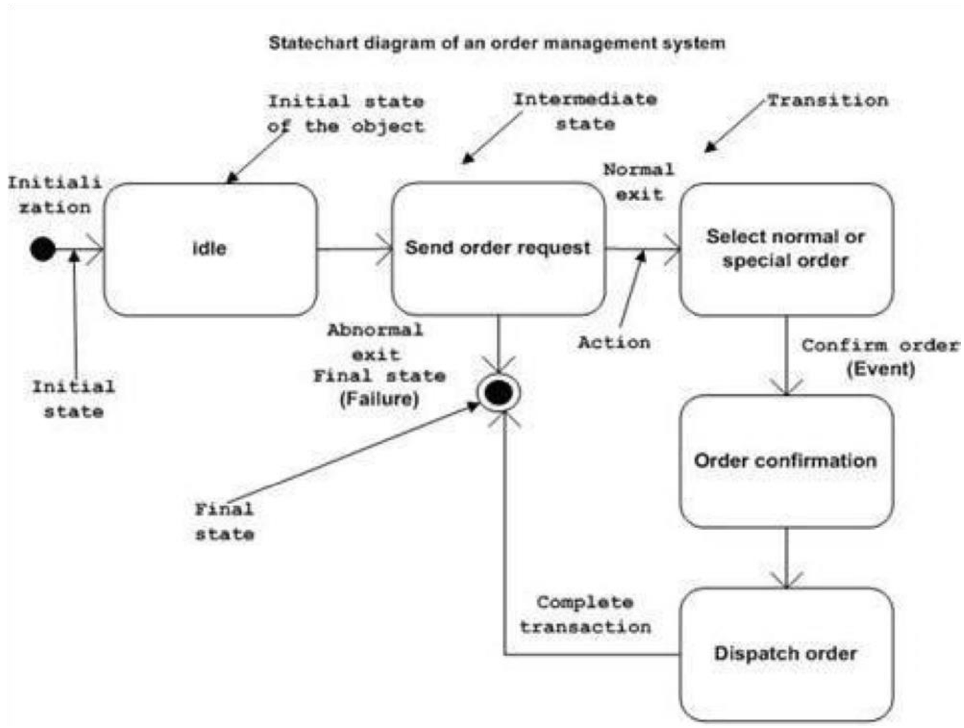
● Identify the events.

The following is an example of a State chart diagram where the state of *Order* object is analyzed.

The first state is an idle state from where the process starts. The next states are arrived for events like *send request*, *confirm request*, and *dispatch order*. These events are responsible for state changes of order object.

During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exists also. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete it is considered as the complete transaction as mentioned below.

The initial and final state of an object is also shown below.



WHERE TO USE STATE CHART DIAGRAMS?

From the above discussion we can define the practical applications of a State chart diagram. State chart diagrams are used to model dynamic aspect of a system like other four diagrams disused in this tutorial. But it has some distinguishing characteristics for modeling dynamic nature.

State chart diagram defines the states of a component and these state changes are dynamic in nature. So its specific purpose is to define state changes triggered by events. Events are internal or external factors influencing the system.

State chart diagrams are used to model states and also events operating on the system. When implementing a system it is very important to clarify different states of an object during its life time and state chart diagrams are used for this purpose. When these states and events are

identified they are used to model it and these models are used during implementation of the system.
If we look into the practical implementation of State chart diagram then it is mainly used to analyze the object states influenced by events. This analysis is helpful to understand the system behavior during its execution.

So the main usages can be described as:

To model object states of a system.

To model reactive system. Reactive system consists of reactive objects.

To identify events responsible for state changes.

Forward and reverse engineering.

Result:Hence we studied and constructed state chart diagram.

CO2	**PO(1,2,3,4,5,8,9,10,11,12)**
COURSE OUTCOME	**PROGRAM OUTCOME Attained**

Experiment No. 9

Aim: To construct the component diagram for hotel management system.

Description:

Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.

Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.

Purpose of Component Diagrams

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

Thus from that point of view, component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc.

Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment. A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole.

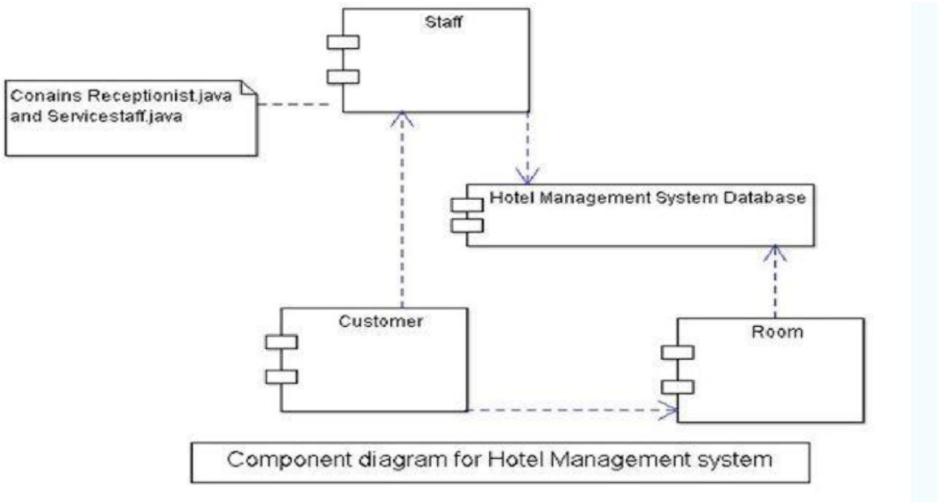
How to Draw a Component Diagram?

Component diagrams are used to describe the physical artifacts of a system. This artifact includes files, executables, libraries, etc

The purpose of this diagram is different. Component diagrams are used during the implementation phase of an application. However, it is prepared well in advance to visualize the implementation details.

Initially, the system is designed using different UML diagrams and then when the artifacts are ready, component diagrams are used to get an idea of the implementation.

This diagram is very important as without it the application cannot be implemented efficiently. A well-prepared component diagram is also important for other aspects such as application performance, maintenance, etc.



Result: Thus we have created component diagram for hotel management system

Conclusion: COURSE OUTCOME	**PROGRAM OUTCOME Attained**
CO4	**PO(1,2,3,4,5,8,9,10,11,12)**

Experiment No: 10

Aim: To construct the deployment for hospital Management System.

Description:

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed.

Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

Purpose of Deployment Diagrams

The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related.

Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components. Most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on the hardware topology of a system. Deployment diagrams are used by the system engineers.

How to Draw a Deployment Diagram?

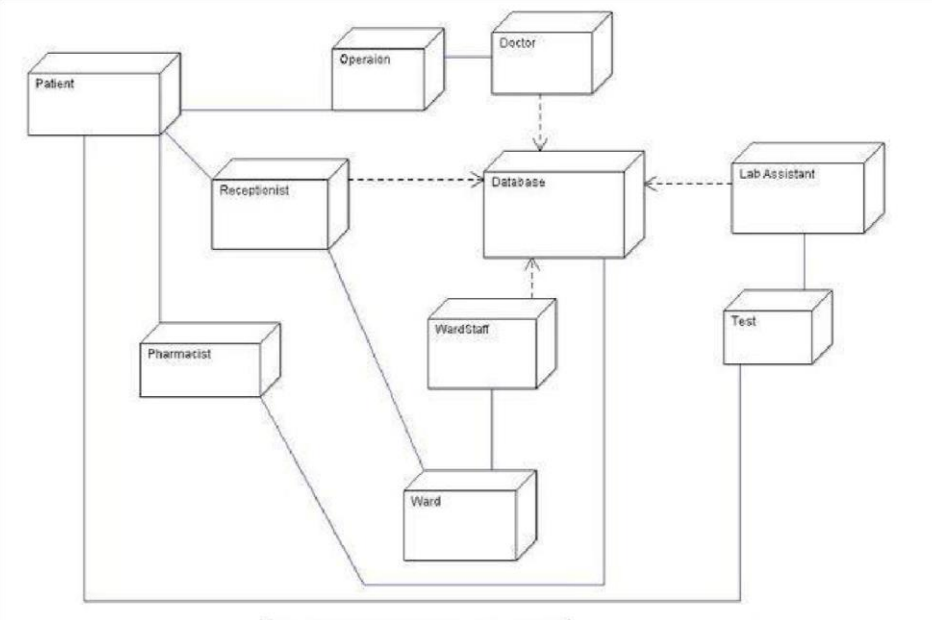
Deployment diagram represents the deployment view of a system. It is related to the component diagram because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application.

Deployment diagrams are useful for system engineers. An efficient deployment diagram is very important as it controls the following parameters −

- Performance
- Scalability
- Maintainability
- Portability

Before drawing a deployment diagram, the following artifacts should be identified −

- Nodes
- Relationships among nodes



Deplyoment diagram for hospital management system

Result: Thus we have created deployment diagram for hospital management system

Conclusion: COURSE OUTCOME	PROGRAM OUTCOME Attained
CO4	PO(1,2,3,4,5,8,9,10,11,12

Experiment No. 11

Aim: To construct the entity relationship diagram for student enrolment system.

DESCRIPTION:

An Entity Relationship Diagram (ERD) is a visual representation of **different entities within a system and how they relate to each other**. For example, the elements writer, novel, and a consumer may be described using ER diagrams the following way:

History of ER Diagrams

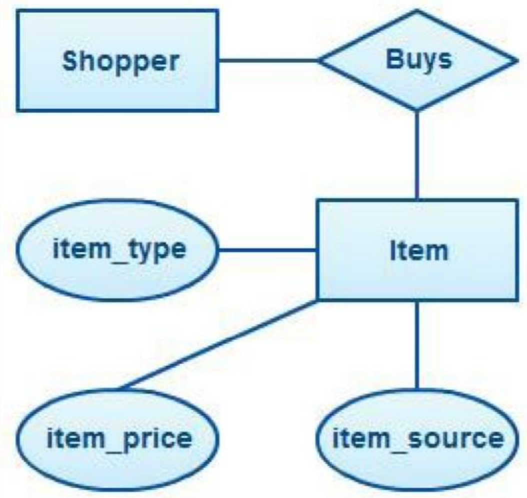
Although data modeling has become a necessity around 1970's there was no standard way to model databases or business processes. Although many solutions were proposed and discussed none were widely adopted.

Peter Chen is credited with introducing the widely adopted ER model in his paper "The Entity Relationship Model-Toward a Unified View of Data". The focus was on entities and relationships and he introduced a diagramming representation for database design as well. His model was inspired by the data structure diagrams introduced by Charles Bachman. One of the early forms of ER diagrams, Bachman diagrams are named after him.

Entity relationship diagrams in software engineering

Entity relationship diagrams are used in software engineering during the planning stages of the software project. They help to identify different system elements and their relationships with each other. It is often used as the basis for data flow diagrams or DFD's as they are commonly known.

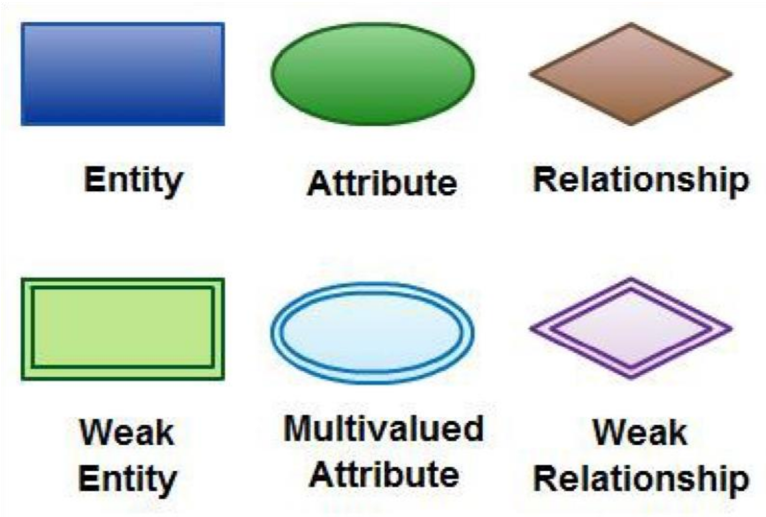
For example, an inventory software used in a retail shop will have a database that monitors elements such as purchases, item, item type, item source and item price. Rendering this information through an ER diagram would be something like this:



ER diagram example with entity having attributes

In the diagram, the information inside the oval shapes are attributes of a particular entity.

Entity Relationship Diagram (ERD) Symbols and Notations Elements in



Elements in ER diagrams

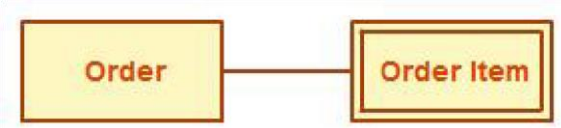
There are three basic elements in an ER Diagram: entity, attribute, relationship. There are more elements which are based on the main elements. They are weak entity, multi valued attribute, derived attribute, weak relationship, and recursive relationship. Cardinality and ordinality are two other notations used in ER diagrams to further define relationships.

Entity

An entity can be a person, place, event, or object that is relevant to a given system. For example, a school system may include students, teachers, major courses, subjects, fees, and other items. Entities are represented in ER diagrams by a rectangle and named using singular nouns.

Weak Entity

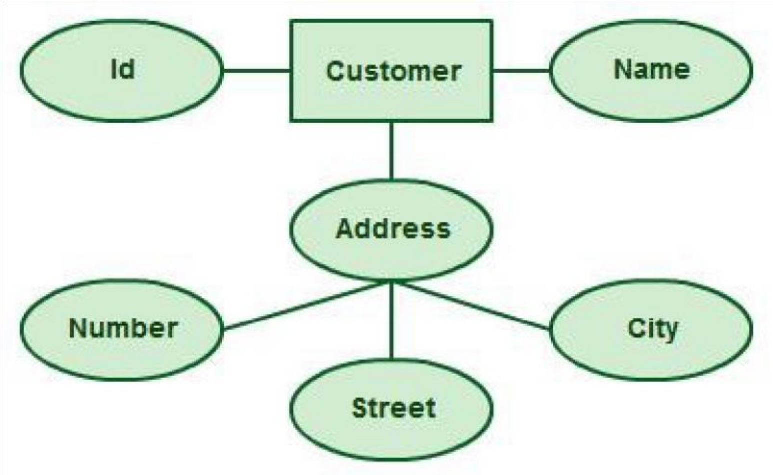
A weak entity is an entity that depends on the existence of another entity. In more technical terms it can be defined as an entity that cannot be identified by its own attributes. It uses a foreign key combined with its attributed to form the primary key. An entity like order item is a good example for this. The order item will be meaningless without an order so it depends on the existence of the order.



Weak Entity Example in ER diagrams

Attribute

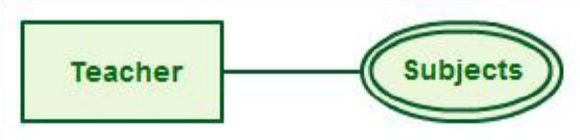
An attribute is a property, trait, or characteristic of an entity, relationship, or another attribute. For example, the attribute Inventory Item Name is an attribute of the entity Inventory Item. An entity can have as many attributes as necessary. Meanwhile, attributes can also have their own specific attributes. For example, the attribute “customer address” can have the attributes number, street, city, and state. These are called composite attributes. Note that some top level ER diagrams do not show attributes for the sake of simplicity. In those that do, however, attributes are represented by oval shapes.



Attributes in ER diagrams, Note that an attribute can have its own attributes (composite attribute)

Multivalued Attribute

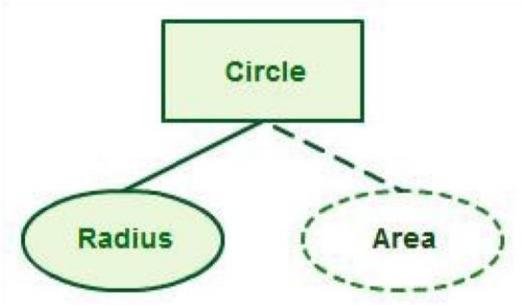
If an attribute can have more than one value it is called a multi-valued attribute. It is important to note that this is different from an attribute having its own attributes. For example, a teacher entity can have multiple subject values.



Example of a multivalued attribute

Derived Attribute

An attribute based on another attribute. This is found rarely in ER diagrams. For example, for a circle, the area can be derived from the radius.



Derived Attribute in ER diarams

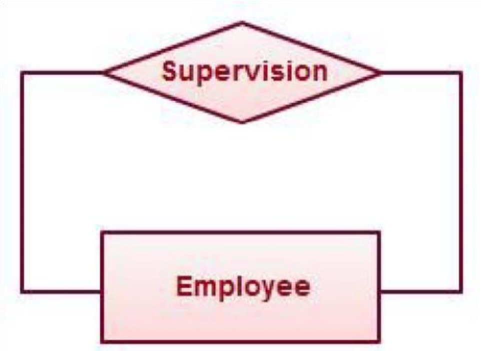
Relationship

A relationship describes how entities interact. For example, the entity “Carpenter” may be related to the entity “table” by the relationship “builds” or “makes”. Relationships are represented by diamond shapes and are labeled using verbs.

Using Relationships in Entity Relationship Diagrams

Recursive Relationship

If the same entity participates more than once in a relationship it is known as a recursive relationship. In the below example an employee can be a supervisor and be supervised, so there is a recursive relationship.



Example of a recursive relationship in ER diagrams

Cardinality and Ordinality

These two further defines relationships between entities by placing the relationship in the context of numbers. In an email system, for example, one account can have multiple contacts. The relationship, in this case, follows a “one to many” model. There are a number of notations used to present cardinality in ER diagrams. Chen, UML, Crow’s foot, Bachman are some of the popular notations. Creately supports Chen, UML and Crow’s foot notations.The following example uses UML to show cardinality.

Cardinality in ER diagrams using UML notation

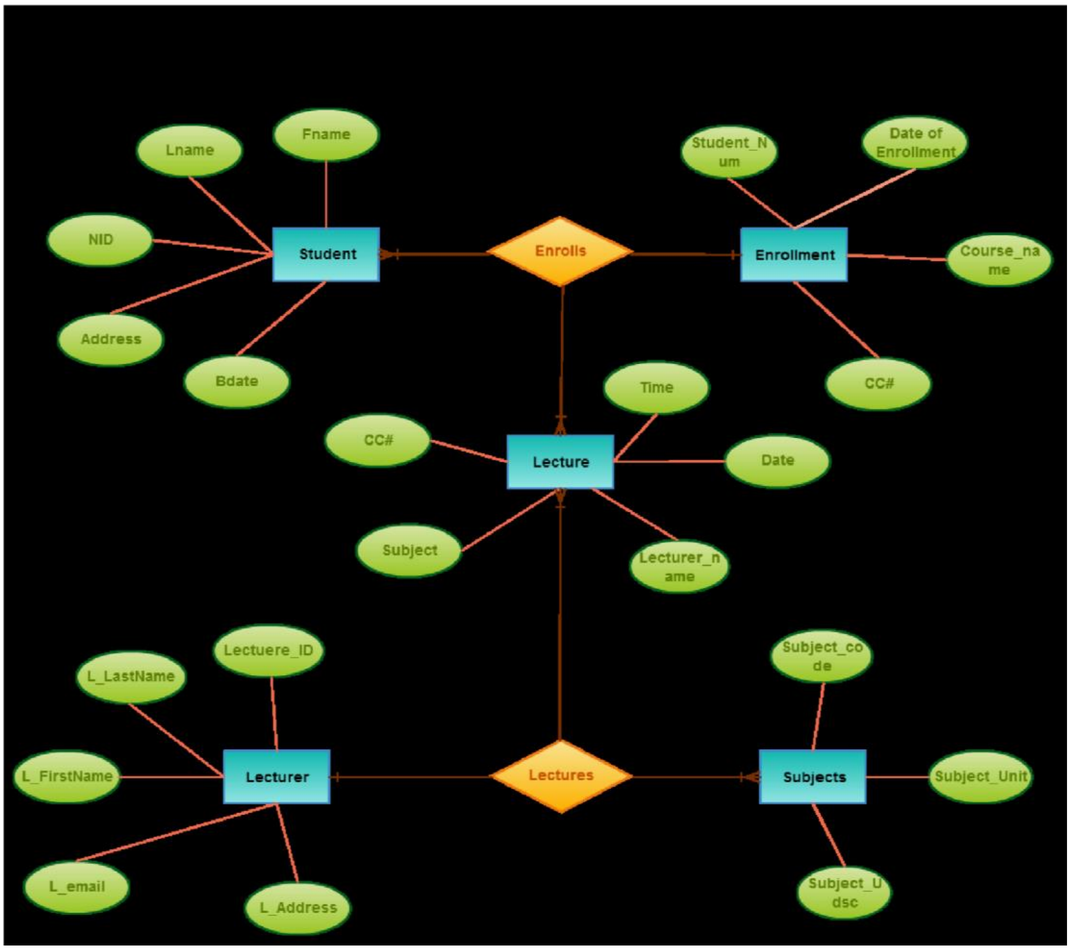
How to Draw ER Diagrams

Below points show how to go about creating an ER diagram.

1. **Identify all the entities** in the system. An entity should appear only once in a particular diagram. Create rectangles for all entities and name them properly.

2. **Identify relationships** between entities. Connect them using a line and add a diamond in the middle describing the relationship.

3. **Add attributes** for entities. Give meaningful attribute names so they can be understood easily.



Result: Thus we have created ER(Entity Relationship)diagram for student enrolment system Conclusion:

COURSE OUTCOME	**PROGRAM OUTCOME Attained**
CO4	**PO(1,2,3,4,5,8,9,10,11,12)**