***Use Case Diagram***

Used for describing a set of user **scenarios** to capture user requirements. Work like a **contract** between end user and software developers

Created to visualize the relationships between **actors** and **use cases** in a scenario *describing how the system is used.*

Use case analysis identifies functional requirements for a scenario

**A functional requirement** describes specific behavior or functions of a system and what a **system** is supposed to accomplish.

**N.B Non-functional requirement** specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.

Elaborates a performance characteristic of the system.

Examples of non-functional requirements

* Accessibility
* Speed
* Disaster recovery
* Efficiency
* Effectiveness
* Fault tolerance

***An actor***is someone or something that must interact with the system under development

An actor specifies a role played by a user or any other system that interacts with the subject.

**Types of Actors**

1. Primary actor – The stakeholder that calls on the system to deliver one of its services.

It has a goal with respect to the system – one that can be satisfied by its operation.

The primary actor is often, but not always, the actor who triggers the use case.

1. Secondary Actors: Actors that the system needs assistance from to achieve the primary actor’s goal.
2. Supporting actor – involved in dialogue, provide services or information

An external actor that provides a service to the system under design. It might be a high-speed printer, a web service,

1. Off-stage actor – has an interest in the use case

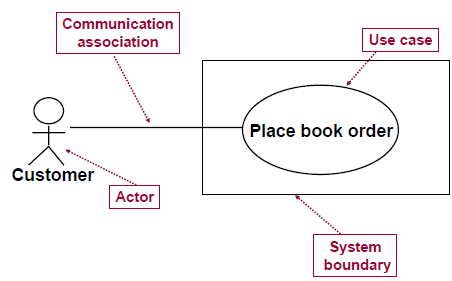
**Example**

1. A bank loan officer wants to review a loan application from a customer, and part of the process involves a real-time credit rating check.
   * Use Case Name: Review Loan Application
   * Primary Actor: Loan Officer
   * Secondary Actors: Credit Rating System /CRB
2. A Human Resources manager wants to change the job code of an employee, and as part of the process, automatically notify several other departments within the company of the change.
   * Use Case Name: Maintain Job Code
   * Primary Actor: Human Resources Manager
   * Secondary Actors: None
3. A Procurement Manager wants to place a “bid” for some goods using an On-Line Trading Community (B2B version of eBay)
   * Use Case Name: Create Bid
   * Primary Actor: Procurement Manager
   * Secondary Actors: On-Line Trading Community

**A scenario**

A scenario is a specific sequence of actions and interactions between actors and the system; also called a use case instance.

**Notation for Use Case Diagrams**



**Relationships among Use Cases**

i) **Inclusion** <<include>>. Implies mandatory.

Used when a use case adds steps to another use case.

A use-case which will be called as a consequence of invoking another one

Indicates a use case that is used (invoked) by another use case

Arrow goes from base case to included use case.

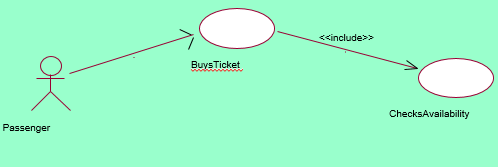
Example



An e-commerce application that provides customers with the option of checking the status of their orders.

This behavior is modeled with a base use case called CheckOrderStatus that has an inclusion use case called LogIn

An include relationship points from the CheckOrderStatus use case to the LogIn use case to indicate that the CheckOrderStatus use case *always includes the behaviors* in the LogIn use case.



Example 2.

**ii) Extension** <<extends>>

Describes extensions to an Existing Use case when one use case adds behaviour to a base case which is executed conditionally.

*N.B : Extend implies Optional* Arrow goes from extended to base use case.

Example



E-commerce system in which you have a base use case called **Place Online Order** that has an extending use case called **Specify Shipping Instructions**.

An extend relationship points from the Specify Shipping Instructions use case to the Place Online Order use case to indicate that the behaviors in the Specify Shipping Instructions use case are *optional and only occur in certain circumstances*.

Extension allows you to create a new use case by adding steps to an existing

***Identifying the Actors***

i. Actor represents the role a user plays with respect to the system.

ii. Think about roles rather than people or job titles.

iii. Who affects the system?

iv. Which user groups are needed by the system to perform its functions? These functions can be both main functions and secondary functions, such as administration.

v. Which external hardware or other systems use the system to perform tasks?

vi. What problems does this application solve

vii. How do users use the system (use case)? What are they doing with the system?

***Guidelines for Finding Use Cases***

• For each actor, find the tasks and functions that the actor should be able to perform or that the system needs the actor to perform.

• Name the use cases.

***Separate Actors From Users***

• Each use case should have only one main actor.

• Isolate users from actors.

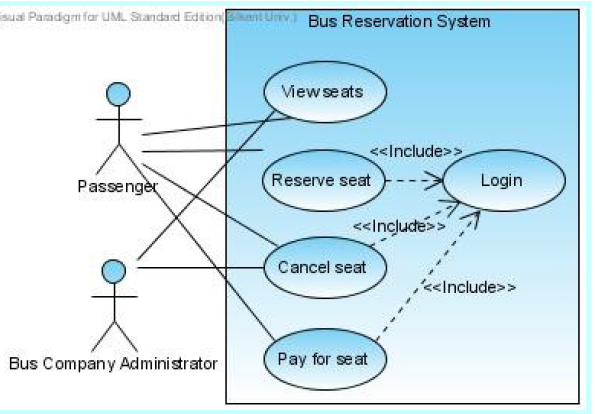
• Isolate actors from other actors (separate the responsibilities of each actor).

• Isolate use cases that have different initiating actors and slightly different behavior.

**Exercise 1**

Seats can be reserved by customers on the web site of the bus company. The customer has the option to directly pay for the seat through the website. In that case, the seat cannot be cancelled (neither by the customer nor by the bus company). If the customer has not paid for the seat, the bus company can cancel the seat if the customer does not show up one hour before the trip. When the reservation is cancelled, the seat will become free and can be sold to another customer. Both the customer and the company staff must authenticate themselves for performing operations with the system.

1. Draw a use case diagram for describing the functional requirements of the above system.



b) List and justify three non-functional requirements that could be important for the above system.

Time Performance: timely response by the system to the passengers is important.

Cost: the new system should not be very expensive; otherwise the budget cannot be justified.

Reuse: it’d be nice for the developer to reuse this system for other reservation systems.

Other possible non-functional requirements include Adaptability and Security.

***Example 2***

The following is a narrative description of the business process with regards to the submitting, reviewing and accepting papers. The author completes an online form that requests the user to input author name, correspondence address, email and, title of paper. The system validates this data and, if correct, asks the author to submit the paper. The author then browses to find the correct paper on their system and submits it. Once received and stored, the system returns to the author a reference number for the paper. Authors may submit as many papers as they like to be considered for acceptance to the conference up until the deadline date for submissions. Papers are allocated to referees for assessment. They review each paper and submit to the system their decision. Once the programme organizer has agreed the decisions authors are informed by email. Accepted papers are then schedule to be delivered at a conference. This involves allocating a date, time and place for the presentation of the paper*.*

Analyse the above text and then draw a use case diagram using an UML modelling tool



**Exercise 3**

Use CASE Scinario

The Northstar Hotel’s website allows potential guests to make a room reservation, specifying the dates and type of room. If they have registered with the website previously their stored details are used to speed up the process, otherwise they are required to register as a new customer. Each reservation is given a unique reservation code.

Before the date of their stay they may enter this reservation code into the website to amend or cancel the reservation. Amendments can include altering the dates, changing the room type or the number of guests in each room.

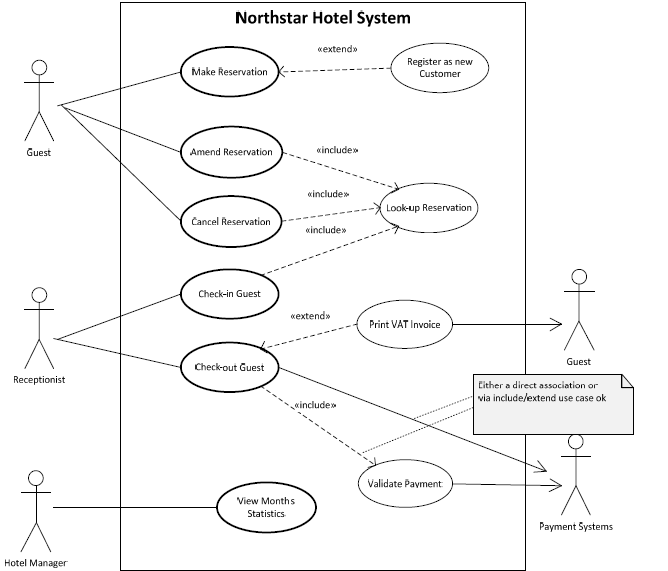
When the guests arrive at the hotel the reservation id is used by the receptionist to quickly find the reservation to check them in with.

At the end of their stay the receptionist checks the guests out, at this point the hotel system validates their payment through the card payment system; a printed invoice may be requested by the guest at this point.

The hotel has many room types available, each with a room-type name, number of guests and additional facility information. Each room in the hotel has a room number and is of one specific type. The maintenance of this data is performed using a separate system out of scope of this exercise. Monthly reports are prepared by the system which may be viewed on request by the

Hotel Manager.

**Required**: Produce a System Use Case diagram for the above scenario.



***Note****: depending on assumptions then some of the extend associations may be represented by include associations, and vice-versa. The only true means of determining this would be to produce full use case descriptions.*

***Class Diagrams (static)***

These are the abstract or physical “things” in a system that model the static view of an application

- A class diagram describes the types of objects in the system and relationships that exist among them.

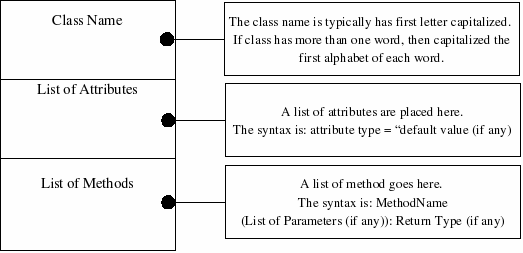
Identifying classes: Find all the nouns and noun phrases in a scenario and consider them as classes

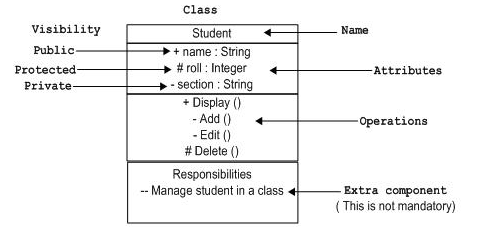
**Purpose of the class diagram**

* Describing the static view of the system.
* Showing the collaboration among the elements of the static view.
* Describing the functionalities performed by the system.
* Construction of software applications using object oriented languages.

Each class is represented by a rectangle subdivided into three compartments

1. Name,
2. Attributes, *Classes have attributes that describe the characteristics of their objects.*
3. Operations (method) **Operations** are actions that each object in the class is responsible for exhibiting or performing ex getFirstName(),





**Access modifiers** /**access specifiers**

Indicate visibility of attributes and operations.

* 1. ‘+’ is used to denote *Public* visibility, usability extends to other classes
  2. ‘#’ is used to denote *Protected* visibility, usability is open only to classes that inherit from original class
  3. ‘-’ is used to denote *Private* visibility, only the original class can use the attribute or operation

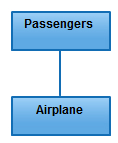
**Relationships in Class Diagrams**

Relationships in class diagrams explains logical connections.

The following are logical connections that are possible in UML:

1. Association
2. Directed Association
3. Reflexive Association
4. Multiplicity
5. Aggregation
6. Composition
7. Inheritance/Generalization
8. Realization

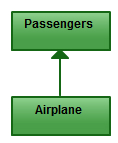
**Association** -- a relationship between instances of the two classes. Logical connection or relationship between classes



Passenger and airline association.

• Example: ―An Employee works for a Company‖

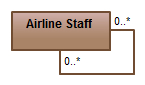
**Directed Association**: refers to a directional relationship represented by a line with an arrowhead. The arrowhead depicts a container-contained directional flow.



### Reflexive Association

Occurs when a class may have multiple functions or responsibilities. For example, a staff working in an airport may be a pilot, aviation engineer, a ticket dispatcher, a guard, or a maintenance crew member.

If the maintenance crew member is managed by the aviation engineer there could be a managed by relationship in two instances of the same class.

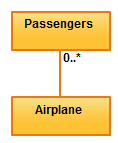


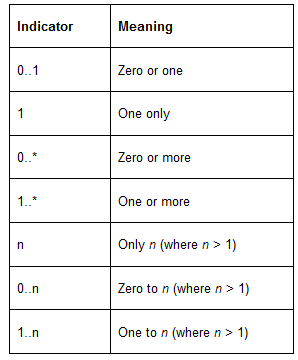
### Multiplicity/ cardinality

Indicate the number of instances of one class linked to one instance of the other class

For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation 0..\* in the diagram means “zero to many”.

For example, one company will have one or more employees, but each employee works for just one company.

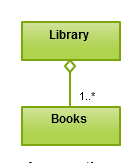


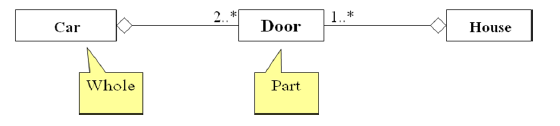


### Aggregation

An association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole.

For example, the class “library” is made up of one or more books, among other materials.



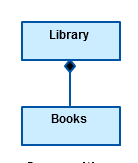
****

### Composition

Similar to the aggregation relationship, with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class.

The contained class will be obliterated when the container class is destroyed. For example, a shoulder bag’s side pocket will also cease to exist once the shoulder bag is destroyed.

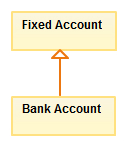
To depict a composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.



### Inheritance / Generalization

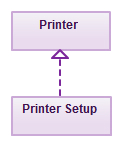
Type of relationship where one associated class is a child of another by assuming the functionalities of the parent class.

To depict inheritance in a UML diagram, a solid line from the child class to the parent class is drawn using an unfilled arrowhead.



### Realization

Denotes the implementation of the functionality defined in one class by another class. To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality to the class that implements the function. In the example, the printing preferences that are set using the printer setup interface are being implemented by the printer.



**Practice Example**

The University of Toronto has several departments. Each department is managed by a chairman, and at least one professor. Professors must be assigned to one, but possibly more departments. At least one professor teaches each course, but a professor may be on sabbatical and not teach any course. Each course may be taught more than once by different professors. We know of the department name, the professor name, the professor employee id, the course names, the course schedule, the term/year that the course is taught, the departments the professor is assigned to, the department that offers the course.

**Identify Classes**

These are the abstract or physical “things” in our system which we wish to describe.

Find all the nouns and noun phrases in the domain descriptions consider them as classes

The class are departments, chairman, professor and course

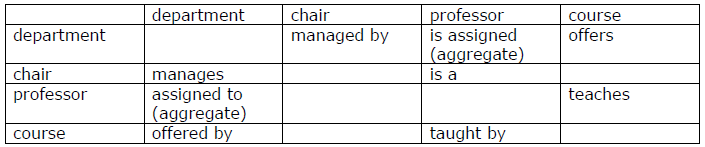
Since there is only one instance of the University of Toronto, we exclude it from our

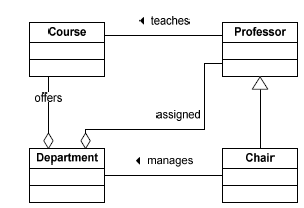
consideration.

**Find Associations**

Find the verbs that join the nouns. e.g., The professor (noun) teaches (verb) students (noun). The verb defines an association between the two nouns.

Identify the type of association. Associations between classes.

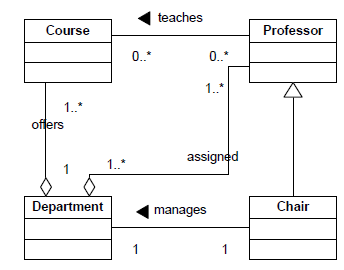
****

****

**Fill in Multiplicity**

Determine the number of occurrences of one class for a single occurrence of the

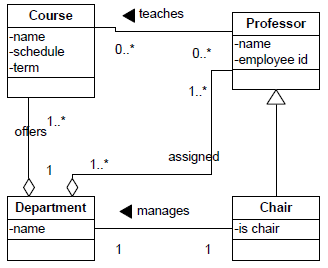
associated class.

****

**Identify Attributes**

Unique identifiers for each class

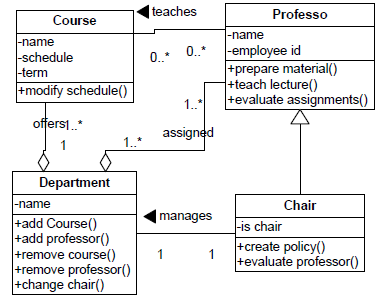
Add these attributes. We have the department name, the professor name, the professor employee id, the course names, the course schedule, the term/year that the course is taught, the departments the professor is assigned to, the department that offers the course.

****

**Identify Behaviours (methods & operations)**

Specify the operations that are required for each class.

In this example we are not given any behaviours, so we will have to make them up. What are some behaviours of these classes?

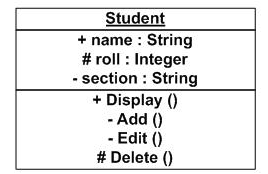
****

## **Object Notation:**

The *object* is represented in the same way as the class. The only difference is the *name* which is underlined.

Object is the actual implementation of a class which is known as the instance of a class.

So it has the same usage as the class.



**Interaction diagram:**

Interactive behavior is represented in UML by two diagrams known as *Sequence diagram* and *Collaboration diagram*. The basic purposes of both the diagrams are similar.

Purposes of interaction diagram can be describes as:

* To capture dynamic behavior of a system.
* To describe the message flow in the system.
* To describe structural organization of the objects.
* To describe interaction among objects.

N.B The following are to identified before drawing the interaction diagram:

* Objects taking part in the interaction.
* Message flows among the objects.
* The sequence in which the messages are flowing.
* Object organization.

**Sequence diagram**: The main purpose of a sequence diagram is to define event sequences that result in some desired outcome.

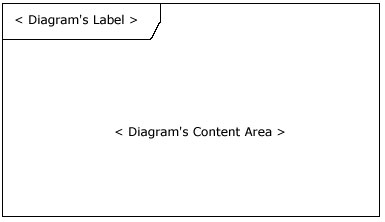
The focus is on the order in which messages occur.

The diagram conveys information along the horizontal and vertical dimensions:

Vertical dimension shows, top down, the time sequence of messages/calls as they occur,

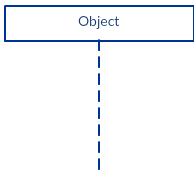
Horizontal dimension shows, left to right, the object instances that the messages are sent to.

**Notations**



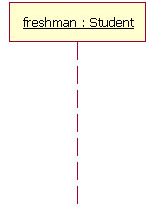
**Lifelines**

Represent the different objects or parts that interact with each other in the system during the sequence.



UML standard for naming a lifeline format

Instance Name: Class Name



The lifeline represents an instance of the class Student, whose instance name is freshman.

### 

### A lifeline notation with an actor element symbol is used when the particular sequence diagram is owned by a use case.

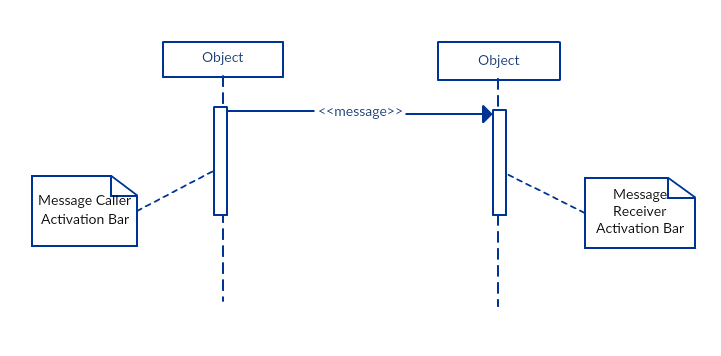
**Activation Bars**

The box placed on the lifeline used to indicate that an object is active during an interaction between two objects.

The length of the rectangle indicates the duration of the objects staying active.

In a sequence diagram, an interaction between two objects occurs when one object sends a message to another.

The use of the activation bar on the lifelines of the Message and the Message Receiver indicates that both are active during the exchange of the message.



**Message Arrows**

An arrow from the Message Caller to the Message Receiver specifies a message in a sequence diagram.

The message arrow comes with a description known as a message signature, on it.

* *Synchronous message*

A synchronous message is used when the sender waits for the receiver to process the message and return before carrying on with another message.

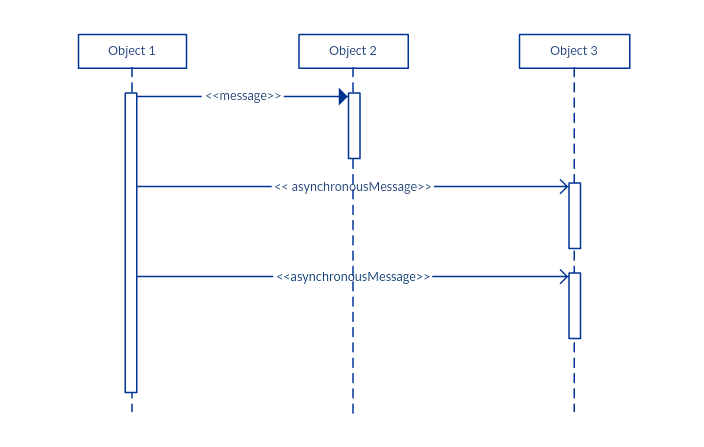
A solid arrow head used to indicate this type of message.



* *Asynchronous message*

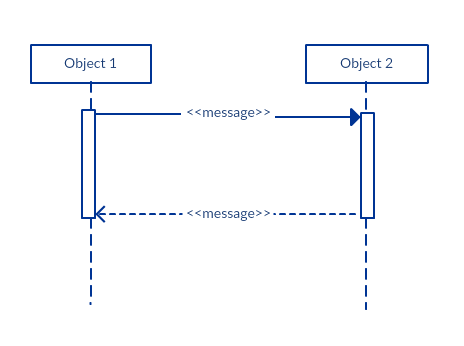
An asynchronous message is used when the message caller does not wait for the receiver to process the message and return before sending other messages to other objects within the system.

A line arrow head is used to show this type of message.



* *Return message*

A return message is used to indicate a return control over to the message caller.

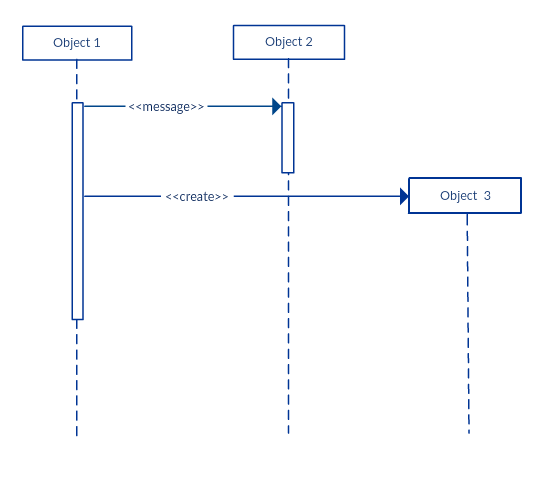


* *Participant  creation message*

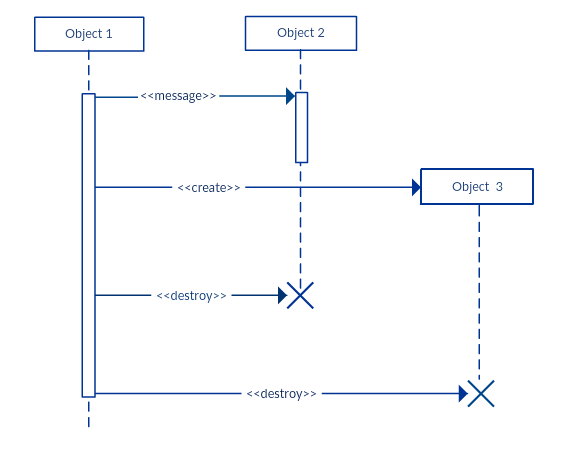
Objects or participants can be created according to the message that is being sent.

A dropped participant box notation is used to show that a particular participant did not exist until

the create call was sent.

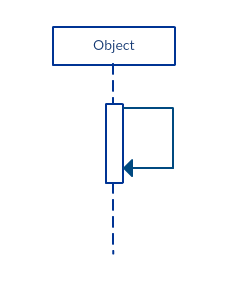


* Participant destruction message

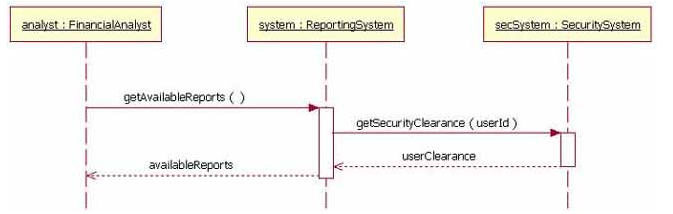
Participants when no longer needed can be deleted from a sequence diagram by adding an ‘X’ at the end of the lifeline of the said part participant.

* *Reflexive message*

When an object sends a message to itself indicated with a message arrow that starts and ends at the same lifeline.



**Examples 1**

****

The analyst object makes a call to the system object which is an instance of the ReportingSystem class.

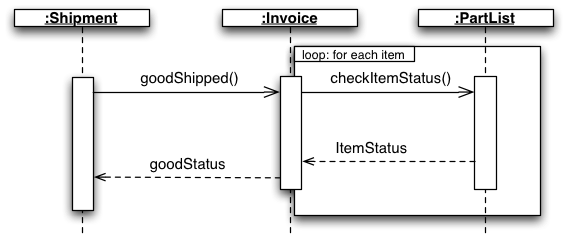
The analyst object is calling the system object's getAvailableReports method.

The system object then calls the getSecurityClearance method with the argument of userId on the secSystem object, which is of the class type SecuritySystem.

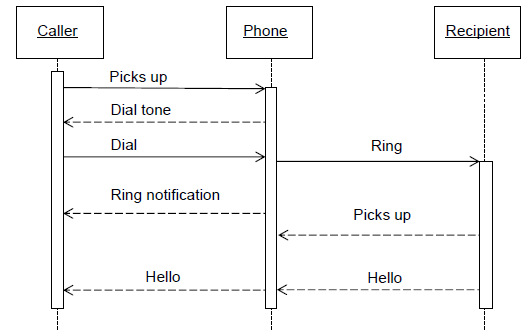
A return message is drawn as a dotted line with an open arrowhead back to the originating lifeline

**Example 2**

The figure below shows an example of a sequence diagram describing 3 objects (instances of classes Shipment, Invoice and PartList)



**Example 3: Sequence Diagram (make a phone call)**

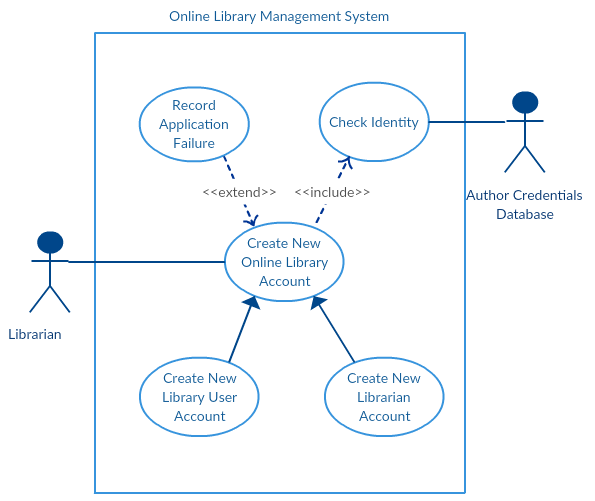


**How to Draw a Sequence Diagram**

A sequence diagram represents the scenario or flow of events in one single use case.

The message flow of the sequence diagram is based on the narrative of the particular use case.

Draw a use case diagram give a comprehensive description of what the particular use case does.



**Exercise 1:** Create New Online Library Account’

From the use case diagram draw a sequence diagram.

**Steps:**

Identify the objects or actors that would be involved in creating a new user account. These would be;

* Librarian
* Online Library Management system
* User credentials database
* Email system

Write a detailed description on what the use case does.

From the description, figure out the interactions (that should go in the sequence diagram) that would occur between the objects once the use case is executed.

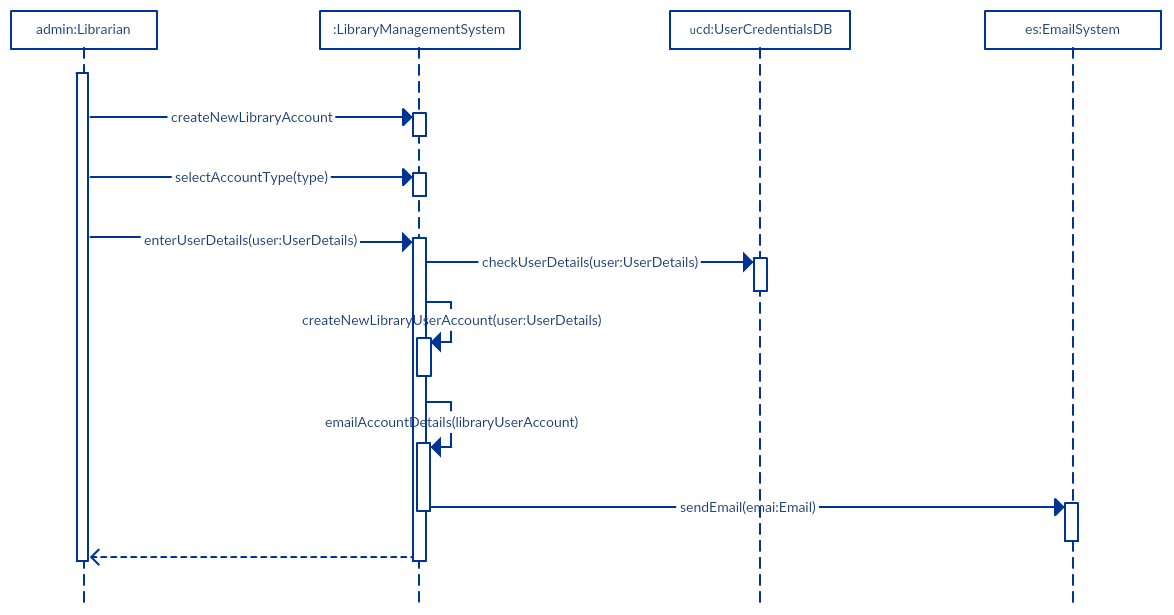
Interactions examples:

* The librarian request the system to create a new online library account
* The librarian then selects the library user account type
* The librarian enters the user’s details
* The user’s details are checked using the user Credentials Database
* The new library user account is created
* A summary of the of the new account’s details are then emailed to the user

From each of the steps, specify what messages should be exchanged between the objects in the sequence diagram.

Once it’s clear draw the sequence diagram.

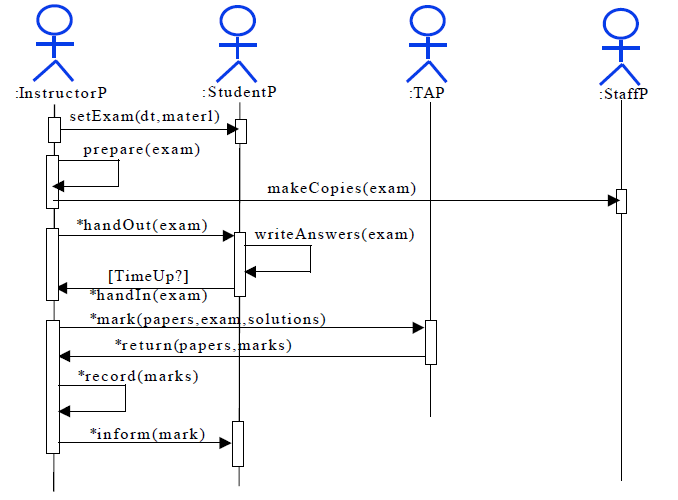
The sequence diagram below shows how the objects in the online library management system interact with each other to perform the function ‘Create New Library User Account’.



**Exercise 2: Instructor sequence diagram**

To give an exam, an instructor first notifies the students of the exam date and the material to be covered. He then prepares the exam paper (with sample solutions), gets it copied to produce enough copies for the class, and hands it out to students on the designated time and location. The students write their answers to exam questions and hand in their papers to the instructor. The instructor then gives the exam papers to the TAs, along with sample solutions to each question, and gets them to mark it. She then records all marks and returns the papers to the students.

Draw a sequence diagram that represents this process. Make sure to show when is each actor participating in the process. Also, show the operation that is carried out during each interaction, and what its arguments are.

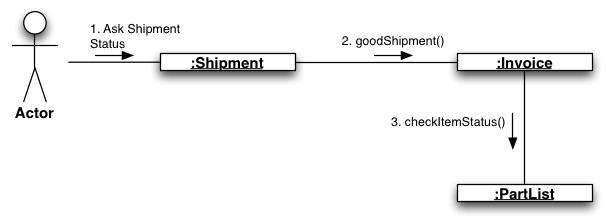


**Collaboration Diagram:** Collaboration diagrams show the message flow between objects and the basic associations (relationships) between classes.

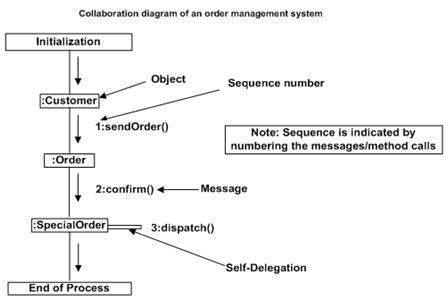
UML Collaboration diagrams illustrate the relationship and interaction between **software objects**.

Collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

In the collaboration diagram the method call sequence is indicated by some numbering technique which indicates how the methods are called one after another.



**Example 2.**



# **UML Component Diagram**

The component diagram's main purpose is to show the structural relationships between the components of a system.

Component diagrams are used to visualize the organization and relationships among components in a system.

A component diagram contains components, interfaces and relationships

* A component diagram provides a physical view of the system. Its purpose is to show the dependencies that the software has on the other software components (e.g., software libraries) in the system. The diagram can be shown at a very high level, with just the large-grain components, or it can be shown at the component package level.

## **Symbols and Notations**

#### Component

A component is a logical unit block of the system, a slightly higher abstraction than classes.

Represented as a rectangle with a smaller rectangle in the upper right corner with tabs or the word <component> written above the name of the component to help distinguish it from a class.

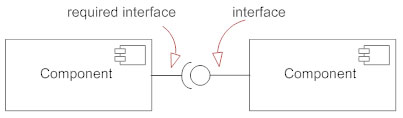


#### Interface

Describes a group of operations used or created by components.

A full circle represents an interface provided by the component.

A semi-circle represents a required interface, like a person's input.

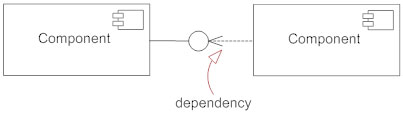


#### Figure 4

#### Order component *provides* two interfaces: OrderEntry and AccountPayable, and the Order component *requires* the Person interface.

#### Dependencies

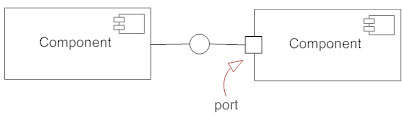
Draw dependencies among components using dashed arrows.



#### Port

Ports are represented using a square along the edge of the system or a component.

A port is often used to help expose required and provided interfaces of a component.



# Figure below, show's the Store's component inner structure.

# Figure 7

# **.**

# **UML Deployment Diagram**

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

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

* Performance
* Scalability
* Maintainability
* Portability

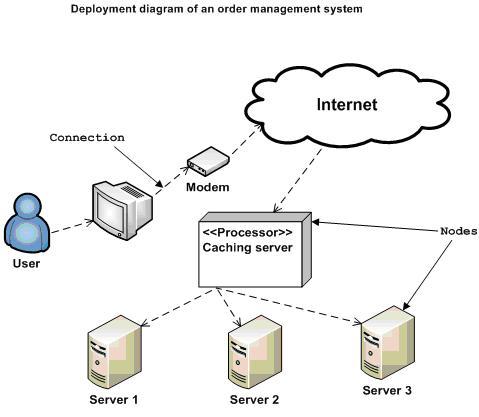
**Example:**

Consider an application assumed to be a web based application which is deployed in a clustered environment using server 1, server 2 and server 3. The user is connecting to the application using internet. The control is flowing from the caching server to the clustered environment. So draw the deployment diagram of order management system.

Here we have shown nodes as:

* Monitor
* Modem
* Caching server
* Server

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



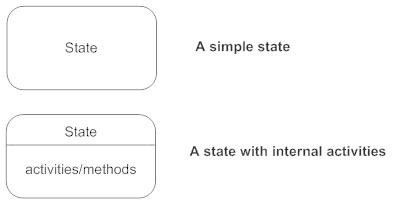
**State Diagrams/ state chart or state machine diagram**

A state diagram shows the behavior of classes in response to external stimuli.

State diagram describes the behavior of a single object in response to a series of events in a system. This UML diagram models the dynamic flow of control from state to state of a particular object within a system.

**Symbols and Notations**

**States**  
States represent situations during the life of an object drawn using a rectangle with rounded corners.



**Transition**  
A solid arrow represents the path between different states of an object.

Label the transition with the event that triggered it and the action that results from it.

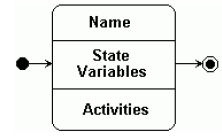
A state can have a transition that points back to itself.



**Initial State**: A filled circle followed by an arrow represents the object's initial state.



**Final State**: An arrow pointing to a filled circle nested inside another circle represents the object's final state.

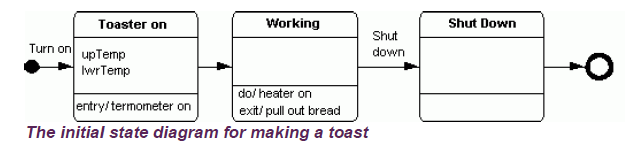
******

**State Diagram – Example**

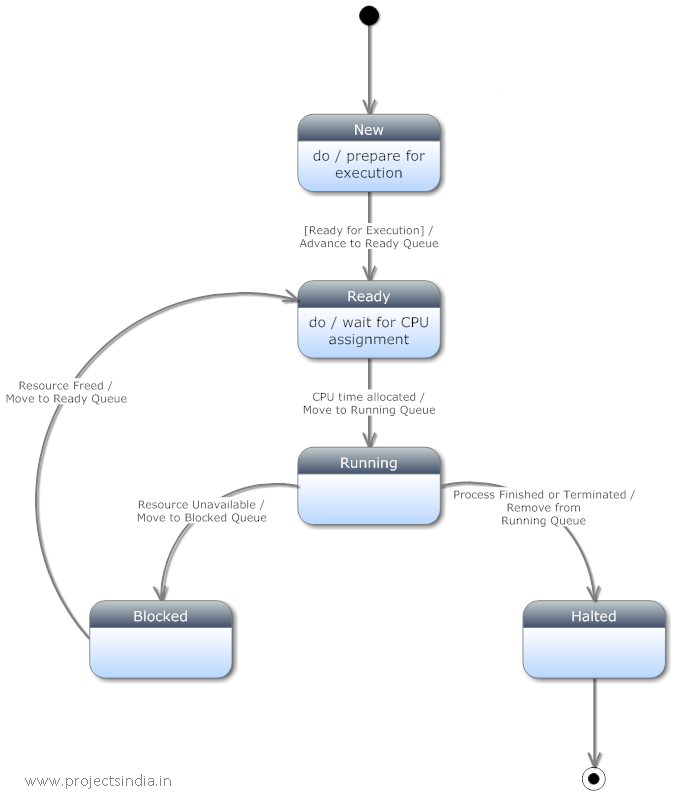
- What are the steps of making a toast?

First of all we must turn on the toaster, put in the bread and wait for several minutes to

bake it. The initial state diagram is shown below:

******

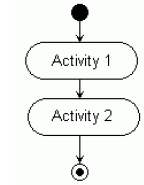
## **State Diagram for CPU Execution**

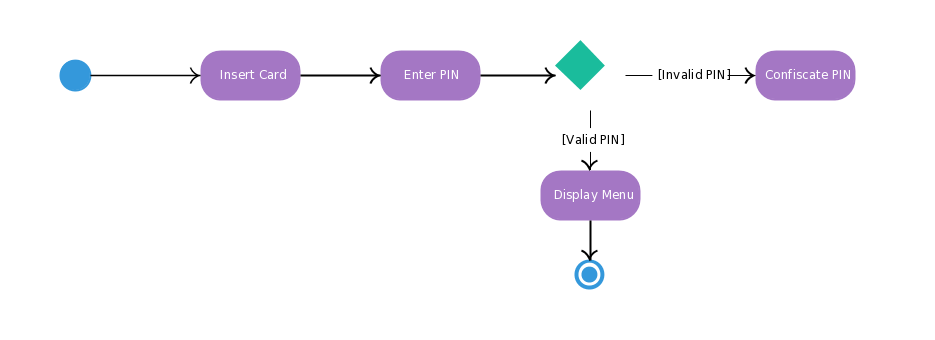


**Activity Diagrams**

Activity diagrams represent workflows in a graphical way.

Sometimes activity diagrams are used as an alternative to State machine diagrams

****



**Draw the activity diagram of ATM Machine system**

