PRACTICAL NO-1

AIM : Implementation of Class Diagram for Hotel management system.

THEORY :

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 shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a *structural diagram*.

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.

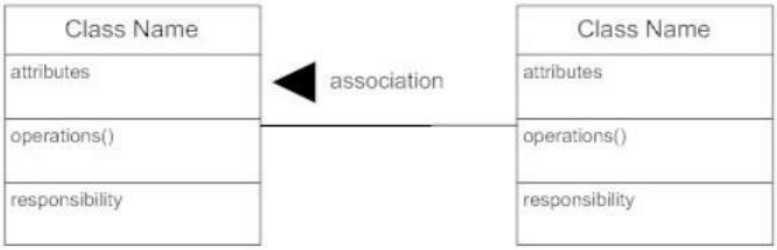
How to draw Class Diagram?

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
- Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

Associations :

Associations represent static relationships between classes. Place association names above, on, or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other.

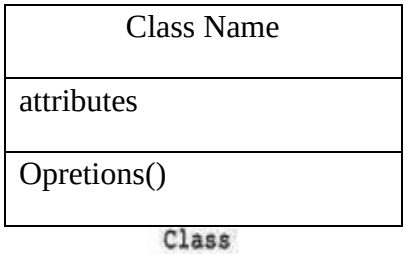


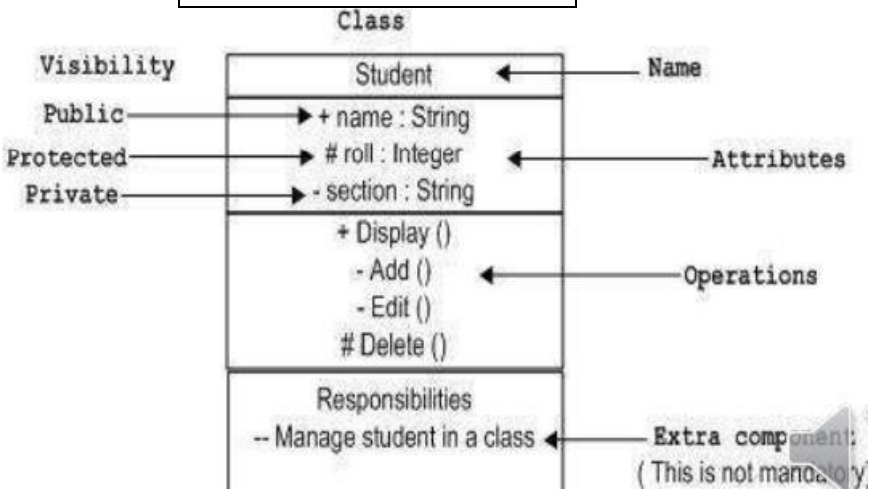
Object :

An object, then, the Animal can have individual objects like cow, lion, dog etc. These are object not only because it has different names, but also because they have different values assigned to their properties.

Class :

A class is a blueprint from which you can create the instance, i.e., objects. A class is used to bind data as well as methods together as a single unit. The class has to be declared only once. Class diagrams provide a graphic notation for modeling classes and their relationships. The UML symbol for a class is a box.



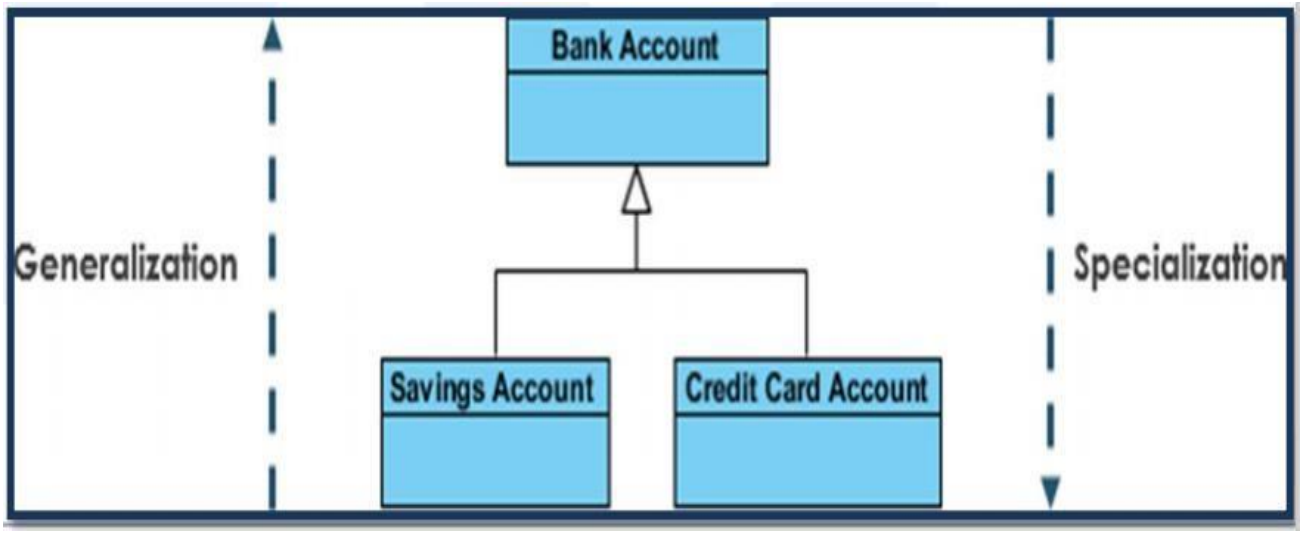


Active classes initiate and control the flow of activity, while passive classes store data and serve other classes. Illustrate active classes with a thicker border.

Generalization & Specialization :

Generalization uses a "is-a" relationship from a specialization to the generalization class. Common structure and behavior are used from the specialization to the generalized class. At a very broader level you can understand this as inheritance. Why I take the term inheritance is, you can relate this term very well. Generalization is also called a "Is-a" relationship.

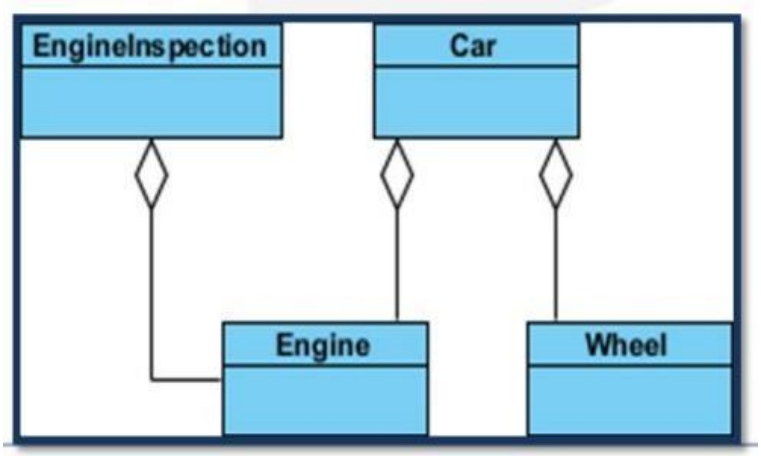
Specialization is the reverse process of Generalization means creating new sub-classes from an existing class. For Example, a Bank Account is of two types - Savings Account and Credit Card Account. Savings Account and Credit Card Account inherit the common/ generalized properties like Account Number, Account Balance, etc. from a Bank Account and also have their specialized properties like unsettled payment etc.



Aggregation :

Aggregation is a special type of association that models a whole- part relationship between aggregate and its parts.

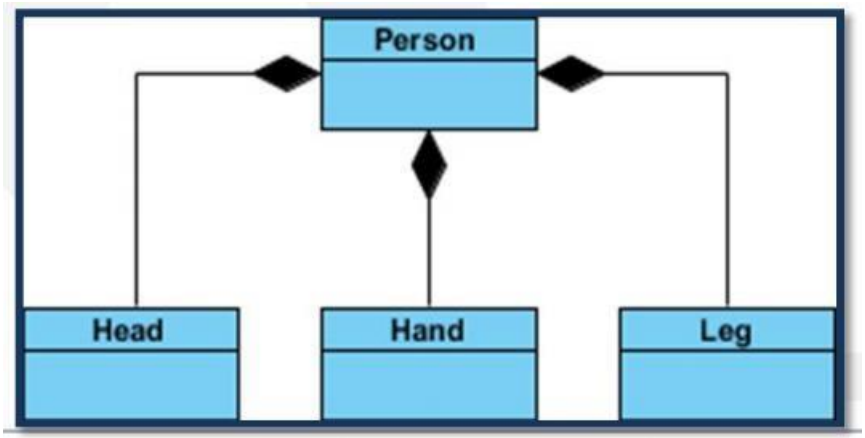
For example, the class college is made up of one or more student. In aggregation, the contained classes are never totally dependent on the lifecycle of the container. Here, the college class will remain even if the student is not available.



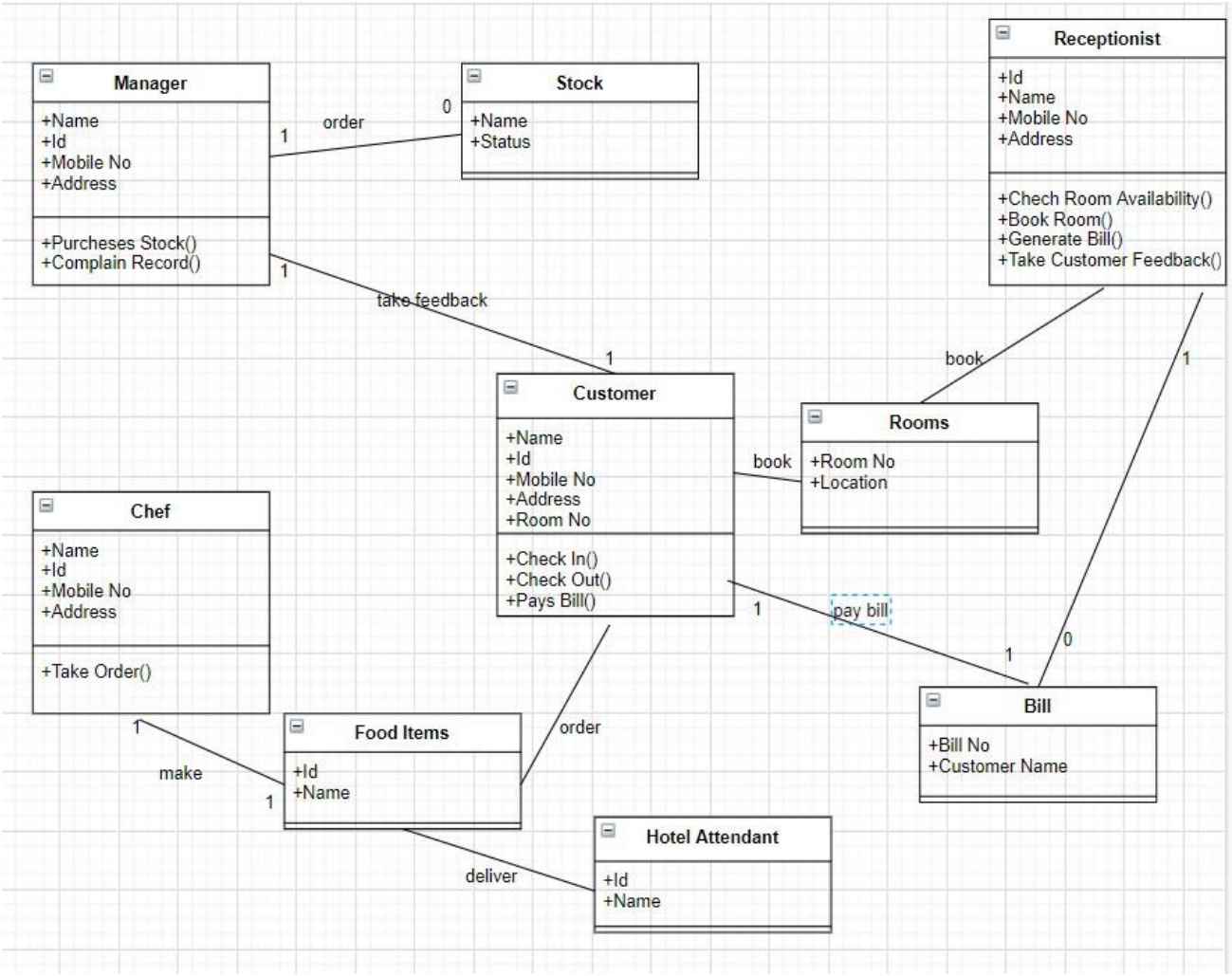
Composition :

The composition is a special type of aggregation which denotes strong ownership between two classes when one class is a part of another class.

For example, if college is composed of classes student. The college could contain many students, while each student belongs to only one college. So, if college is not functioning all the students also removed.



Class Diagram for Hotel management system.



PRACTICAL NO-2

AIM : Implementation of Object Diagram for Order management system.

THEORY :

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.

Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.

Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

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

- Forward and reverse engineering.
- Object relationships of a system
- Static view of an interaction.
- Understand object behaviour and their relationship from practical perspective

How to draw Object Diagram?

- First, analyse the system and decide which instances are having important data and association.
- Second, consider only those instances which will cover the functionality.
- Third, make some optimization as the numbers of instances are unlimited.

It has the following objects

- Customer
- Order
- Specialorder
- NormalOrder

Example:

- Now the customer object (C) is associated with three order objects (O1, O2 and O3). These order objects are associated with special order and normal order objects (S1, S2 and N1). The customer is having the following three orders with different numbers (12, 32 and 40) for the particular time considered.

- Now the customer can increase number of orders in future and in that scenario the object diagram will reflect that. If order, special order and normal order objects are observed then you will find that they are having some values.
- For orders the values are 12, 32, and 40 which implies that the objects are having these values for the particular moment (here the particular time when the purchase is made is considered as the moment) when the instance is captured.
- The same is for special order and normal order objects which are having number of orders as 20, 30 and 60. If a different time of purchase is considered then these values will change accordingly.
- So the following object diagram has been drawn considering all the points mentioned above:

Object diagram shows this relation between the instantiated classes and the defined class, and the relation between these objects in the system. They are be useful to explain smaller portions of your system, when your system class diagram is very complex, and also sometimes modeling recursive relationship in diagram. Before you create a class diagram, you might create an object diagram to discover facts about specific examples of the classifiers that are required.

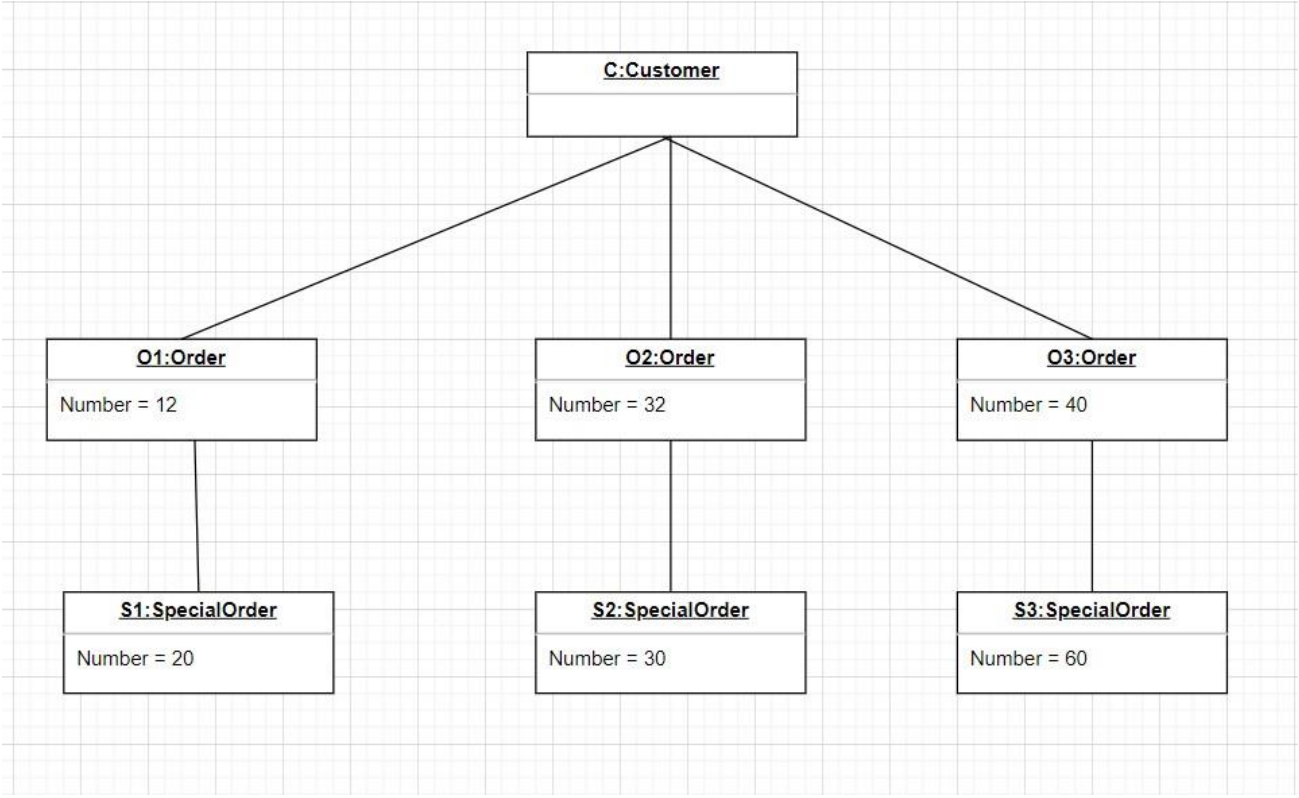
Object Attributes :

Similar to classes, you are able to list object attributes inside a separate compartment. However, unlike classes, object attributes should have values assign for them.

Object Names :

Every object actually symbolized like a rectangle, that offers the name from the object and its class underlined as well as divided with a colon. Obj Name: Class name, ex: Disha:Student.

Object Diagram for Order management system.



PRACTICAL NO-3

AIM : Implementation of State Diagram for Hotel Management system.

THEORY :

Initial state : We use a black filled circle represent the initial state of a system.

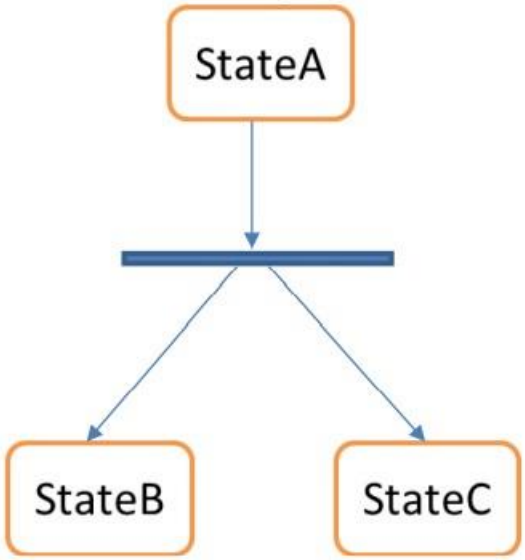
Transition : we use a solid arrow to represent the transition or change of control from one state to another.

Self transition : We use a solid arrow pointing back to the state itself to represent a self transition.
Composite state: We use a rounded rectangle to represent a composite state also

State : We use a rounded rectangle to represent a state.

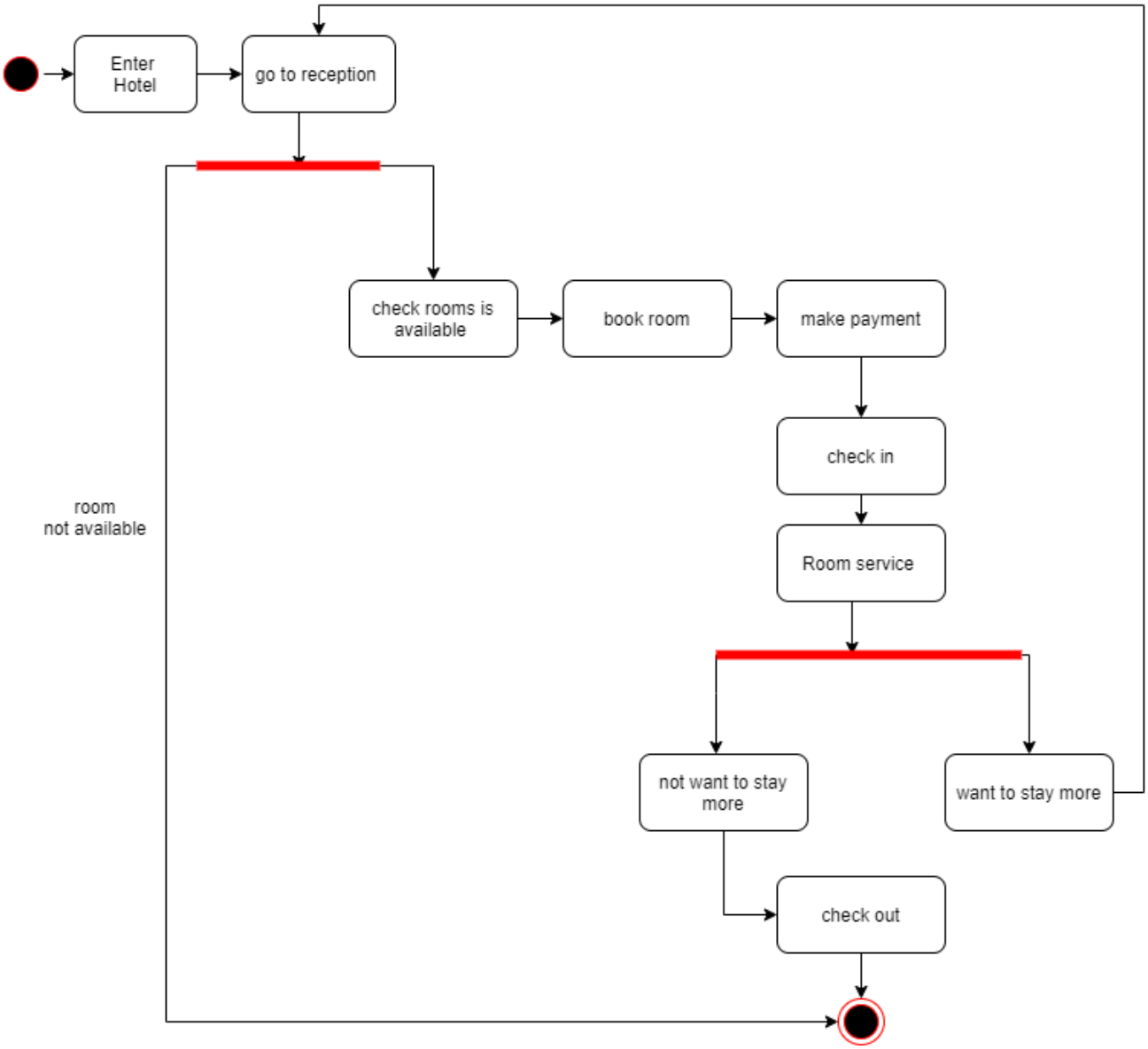
Join : We use a rounded solid rectangular bar to represent a join notation with incoming arrows from the joining state and outgoing arrow towards the common goal state

Fork : we use a rounded solid rectangular bar to represent a fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states.



Final State: we use a filled circle within a circle notation to represent the final state in a state machine diagram.

State Diagram for Hotel Management system.



PRACTICAL NO-4

AIM : Implementation of Use-Case Diagram for Library Management system.

THEORY :

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.

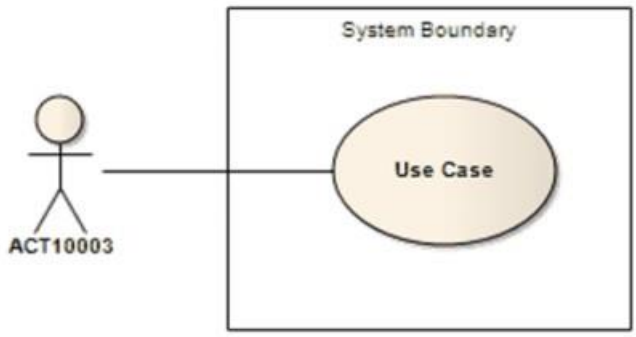
Relationships among the use cases and actors.

Goals:

The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.

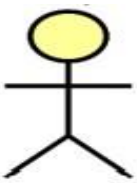
System boundary boxes:

A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system.



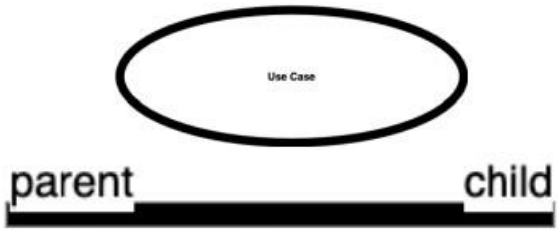
Actors:

Stick figures that represent the people actually employing the use cases.



Use cases:

Horizontally shaped ovals that represent the different uses that a user might have.



System:

A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.

Packages:

A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.

Associations:

A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.

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 analysed the functionalities are captured in use cases.

So, we can say that use 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.

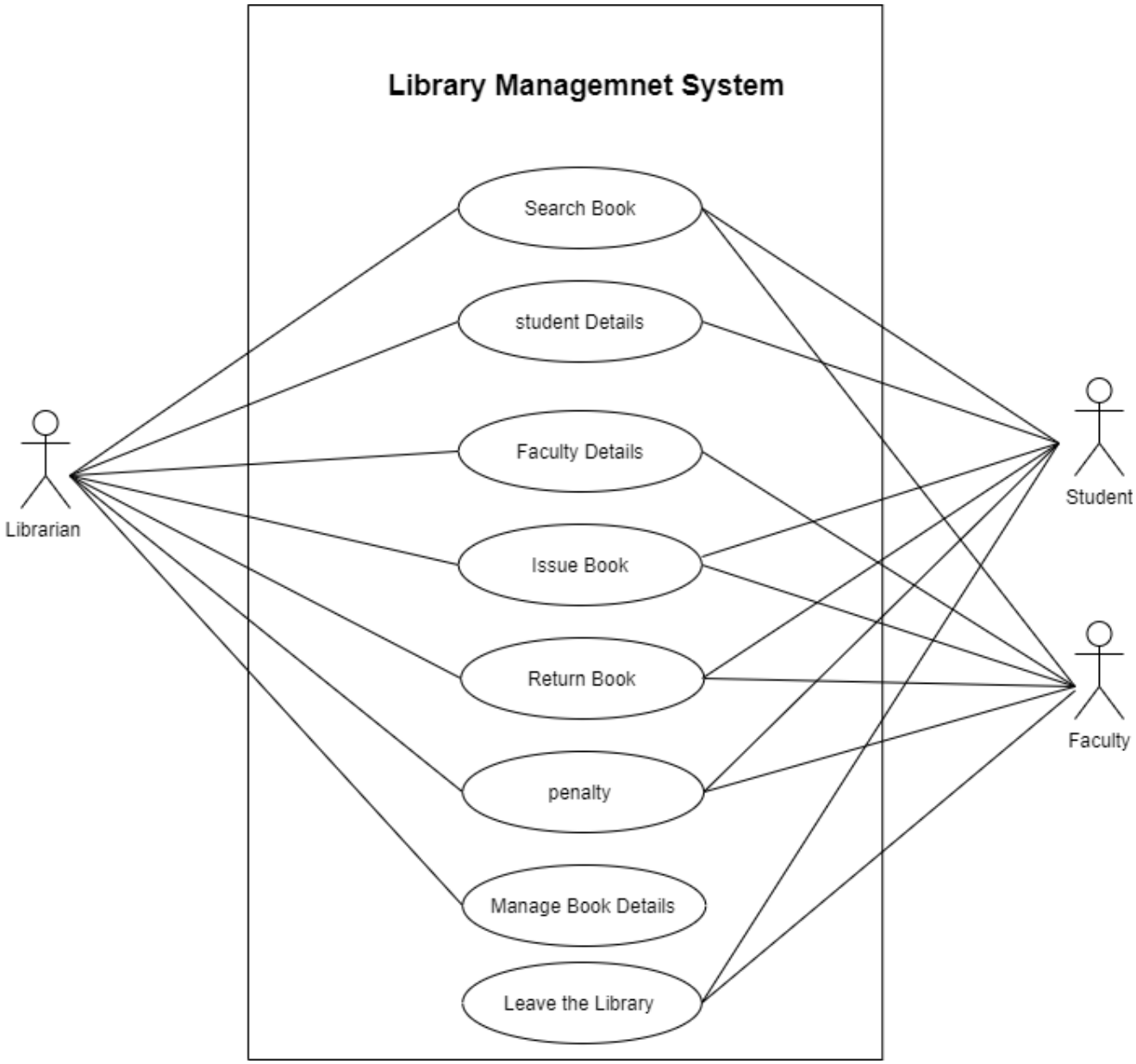
- Functionalities to be represented as a use case
- Actors

- 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 must 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 whenever required to clarify some important points.

Use-Case Diagram for Library Management system.



PRACTICAL NO-5

AIM : Implementation of Sequence Diagram for Hospital management system.

THEORY :

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They're also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios

How to Use Sequence Diagrams

- Model and document how your system will behave in various scenarios
- Validate the logic of complex operations and functions

Basic Sequence Diagram Symbols and Notations

Class Roles or Participants

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

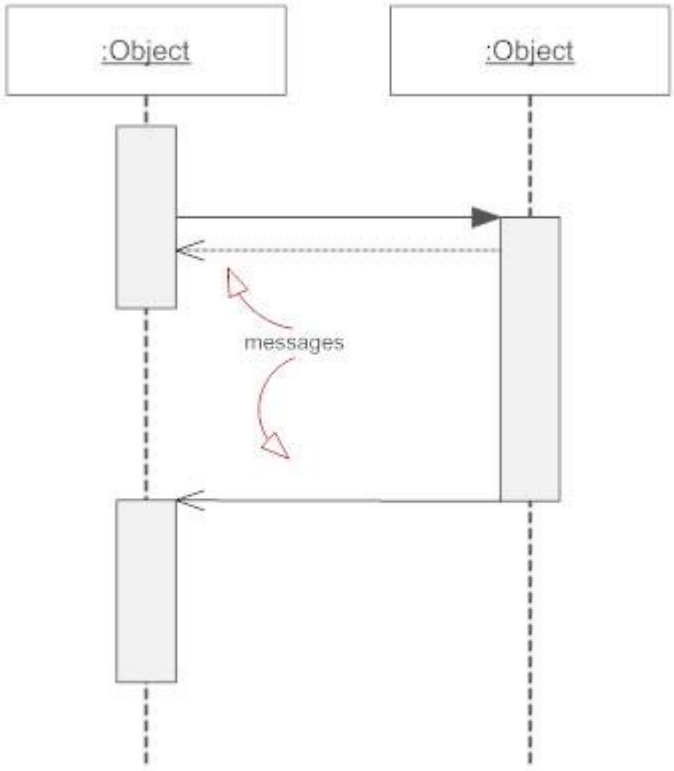
Activation or Execution Occurrence

Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process, or waiting for a reply message, use a thin grey rectangle placed vertically on its lifeline.

Messages

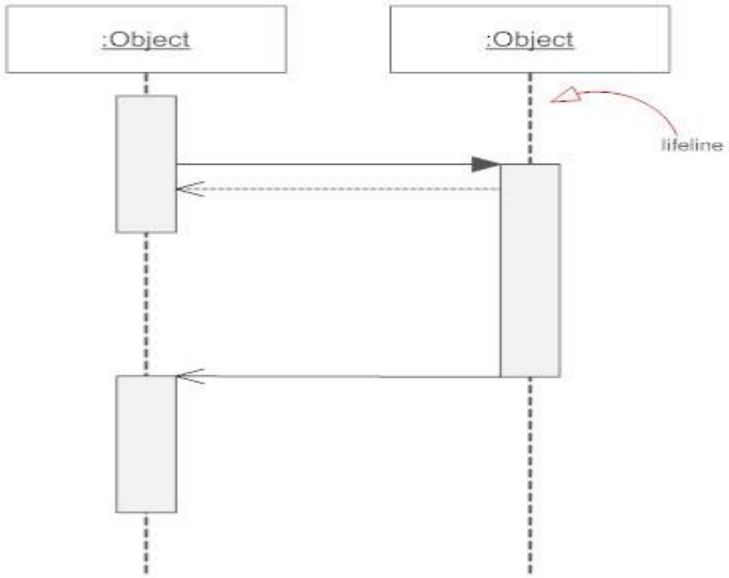
Messages are arrows that represent communication between objects. Use half-arrowed lines to

represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks. For message types, see below.



Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.



Destroying Objects

Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].

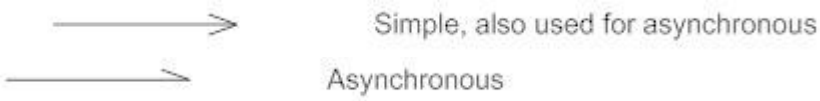
Types of Messages in Sequence Diagrams

Synchronous Message

A synchronous message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to another.

Asynchronous Message

Asynchronous messages don't need a reply for interaction to continue. Like synchronous messages, they are drawn with an arrow connecting two lifelines; however, the arrowhead is usually open and there's no return message depicted.

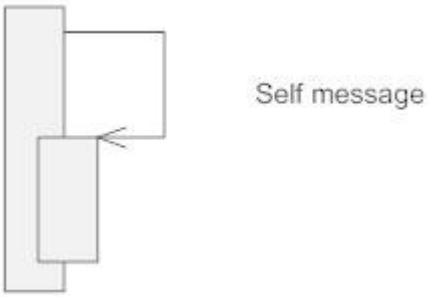


Reply or Return Message

A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

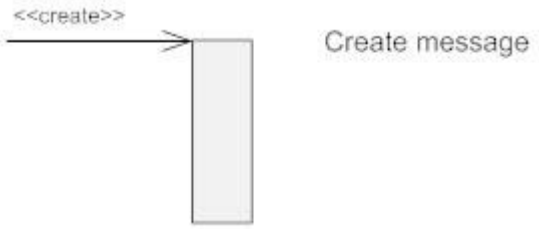
Self Message

A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself.



Create Message

This is a message that creates a new object. Similar to a return message, it's depicted with a dashed line and an open arrowhead that points to the rectangle representing the object created.



Delete Message

This is a message that destroys an object. It can be shown by an arrow with an x at the end.

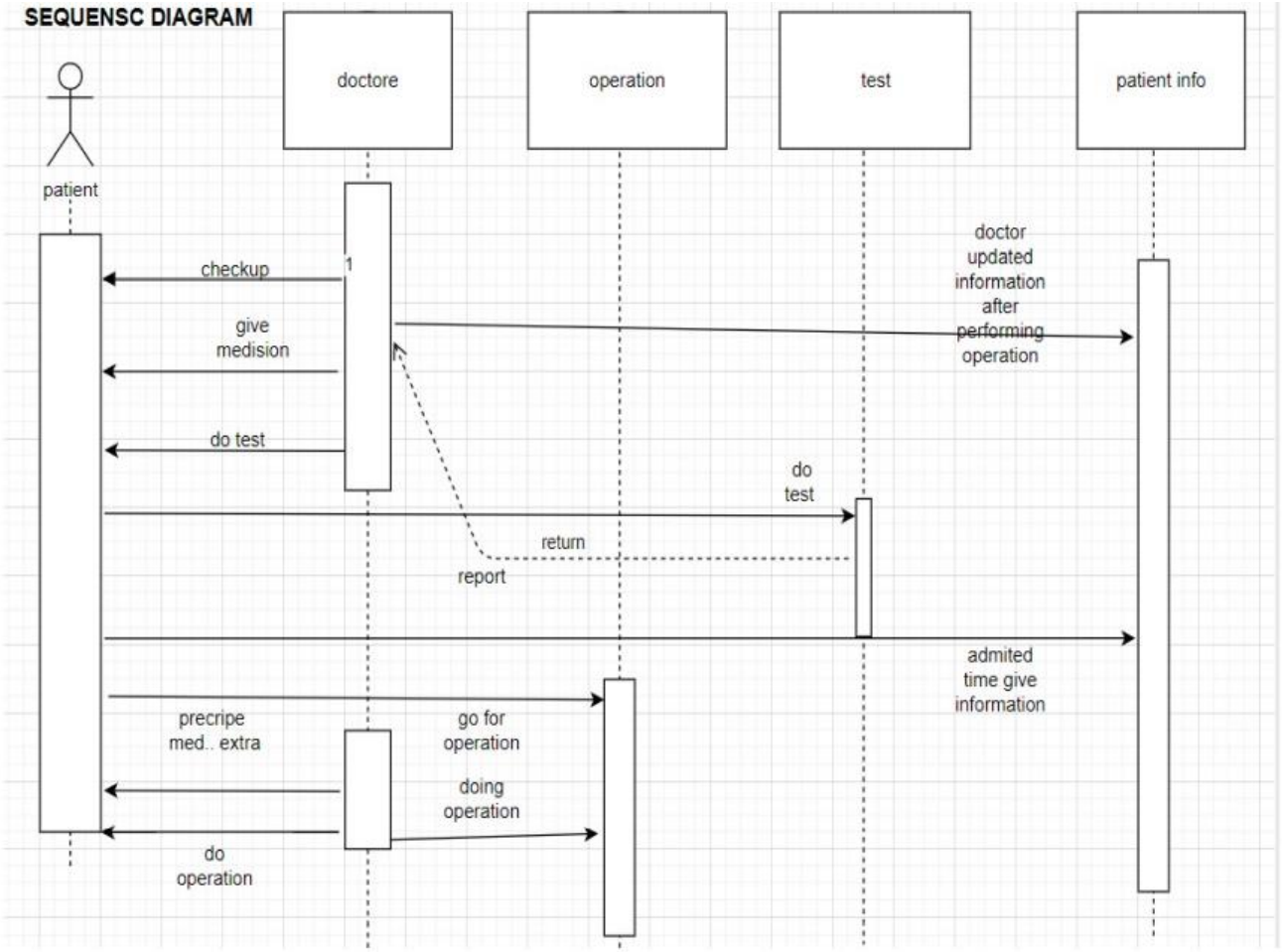
Found Message

A message sent from an unknown recipient, shown by an arrow from an endpoint to a lifeline.

Lost Message

A message sent to an unknown recipient. It's shown by an arrow going from a lifeline to an endpoint, a filled circle or an x.

Sequence Diagram for Hospital management system.



PRACTICAL NO-6

AIM : Implementation of Activity Diagram for Online Banking system.

THEORY :

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.

The purposes of Activity Diagram 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 is not exactly a flow chart as they have some additional capabilities. These additional capabilities include branching, parallel flow, swim lane 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.

Start Point or Initial State:

A small filled circle followed by an arrow represents the initial action state or the start point for any activity diagram. For activity diagram using swimlanes, make sure the start point is placed in the top left corner of the first column.

Start Point/Initial State

Active or Action State :

An action state represents the non-interruptible action of objects. You can draw an action state in SmartDraw using a rectangle with rounded corners.

Activity

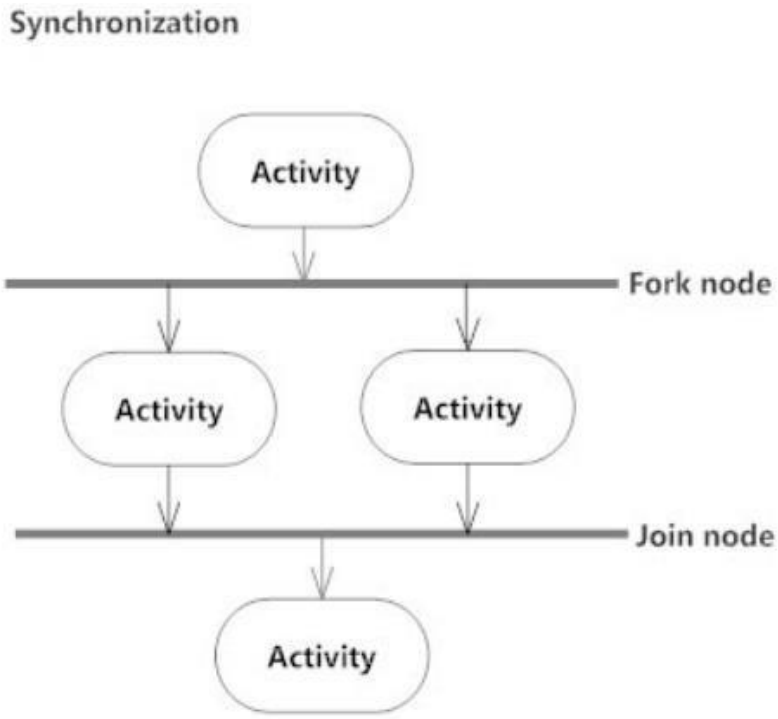
Action Flow :

Action flows, also called edges and paths, illustrate the transitions from one action state to another. They are usually drawn with an arrowed line.

Action Flow

Synchronization :

A fork node is used to split a single incoming flow into multiple concurrent flows. It is represented as a straight, slightly thicker line in an activity diagram. A fork and join mode used together are often referred to as synchronization A join node joins multiple concurrent flows back into a single outgoing flow.

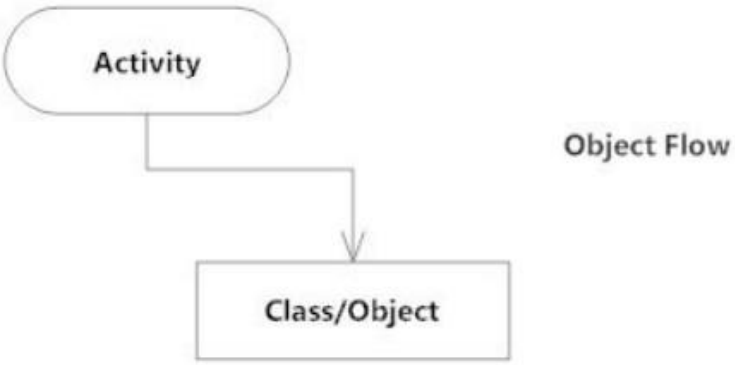


Branching and Decisions :

A diamond represents a decision with alternate paths. When an activity requires a decision prior to moving on to the next activity, add a diamond between the two activities. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."

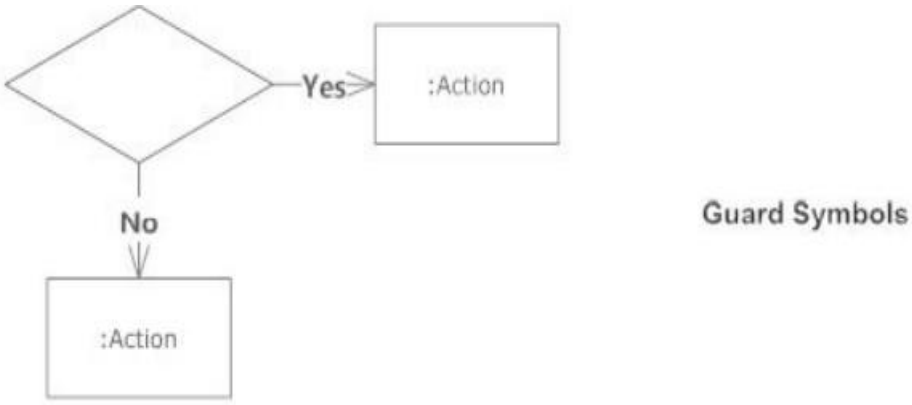
Object Flow :

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



Guards :

In UML, guards are a statement written next to a decision diamond that must be true before moving next to the next activity. These are not essential, but are useful when a specific answer, such as "Yes, three labels are printed," is needed before moving forward.



Activity Diagram for Online Banking system.