## Unit III-oose

### Use-case diagram

Use-case allows us to model and structure the interactions between the users of a system, called **actors**, and the system itself. The use-cases represent a means of analysis of the users' needs and enable us to relate the actions made by a user with the reactions considering of a system. More precisely, a use-case is an abstraction of a set of concrete scenarios carried out by a type of users (Figure 1).

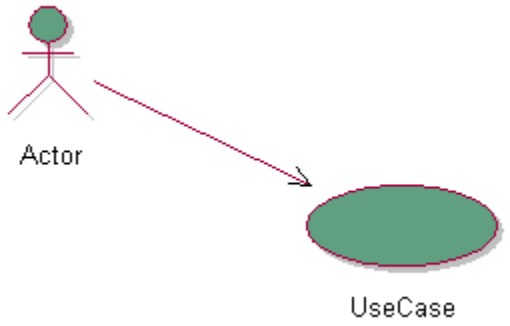



*Figure 1: use-case*

Use-cases can be structured and connected by two principal types of relations: relations of use (**uses**) and relations of extension (**extends**). A **uses** relation allows us to break up a use-case into use sub-cases. A relation of **extension** indicates a specialisation. In Figure 2, the interaction defined by the use-case A includes that of the use-case B, and the use-case C is a particular case of the use-case B.

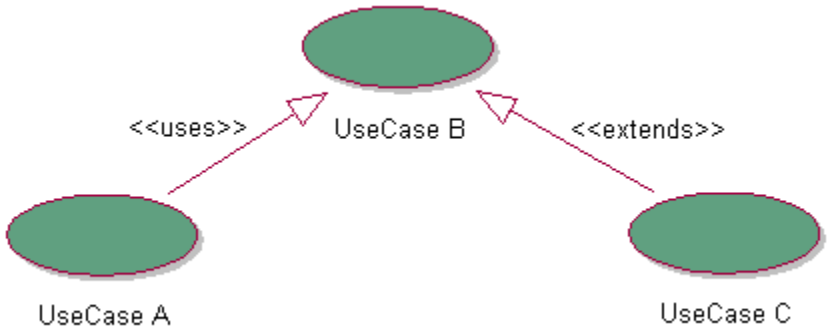



*Figure 2: types of relations between use-cases*

For example (Figure 3), if one wishes to model the operations of credit transfers, they could identify two types of actors: distant customers and local customers. The two types of customers carry out a transfer, but the distant customers use a Minitel terminal, which generates a specific interaction (relation of **extension**). In all the cases, the operation of transfer includes a phase of identification (relation of **use**).

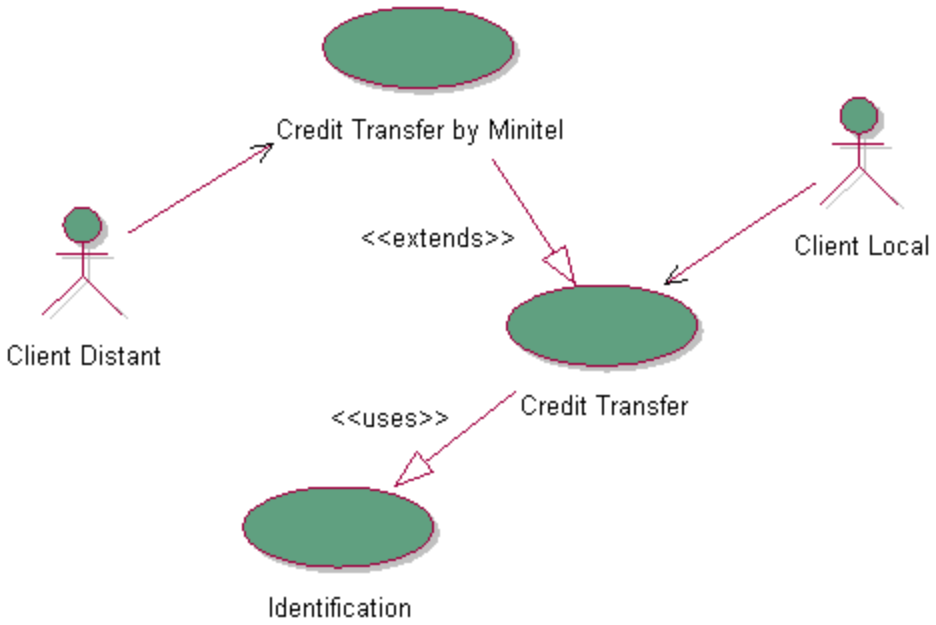
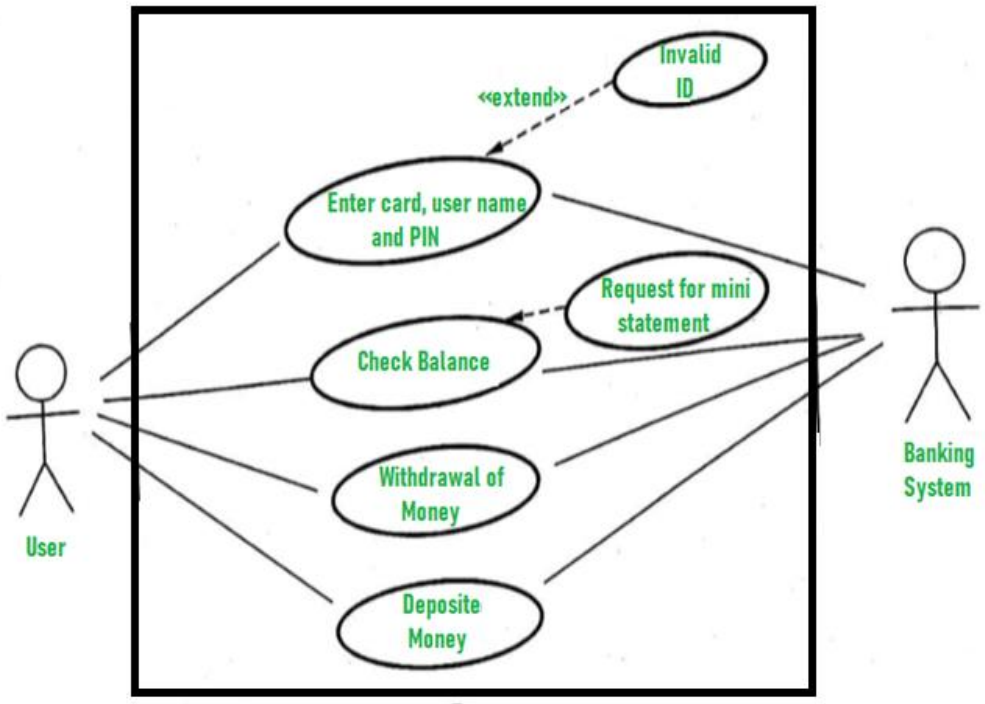
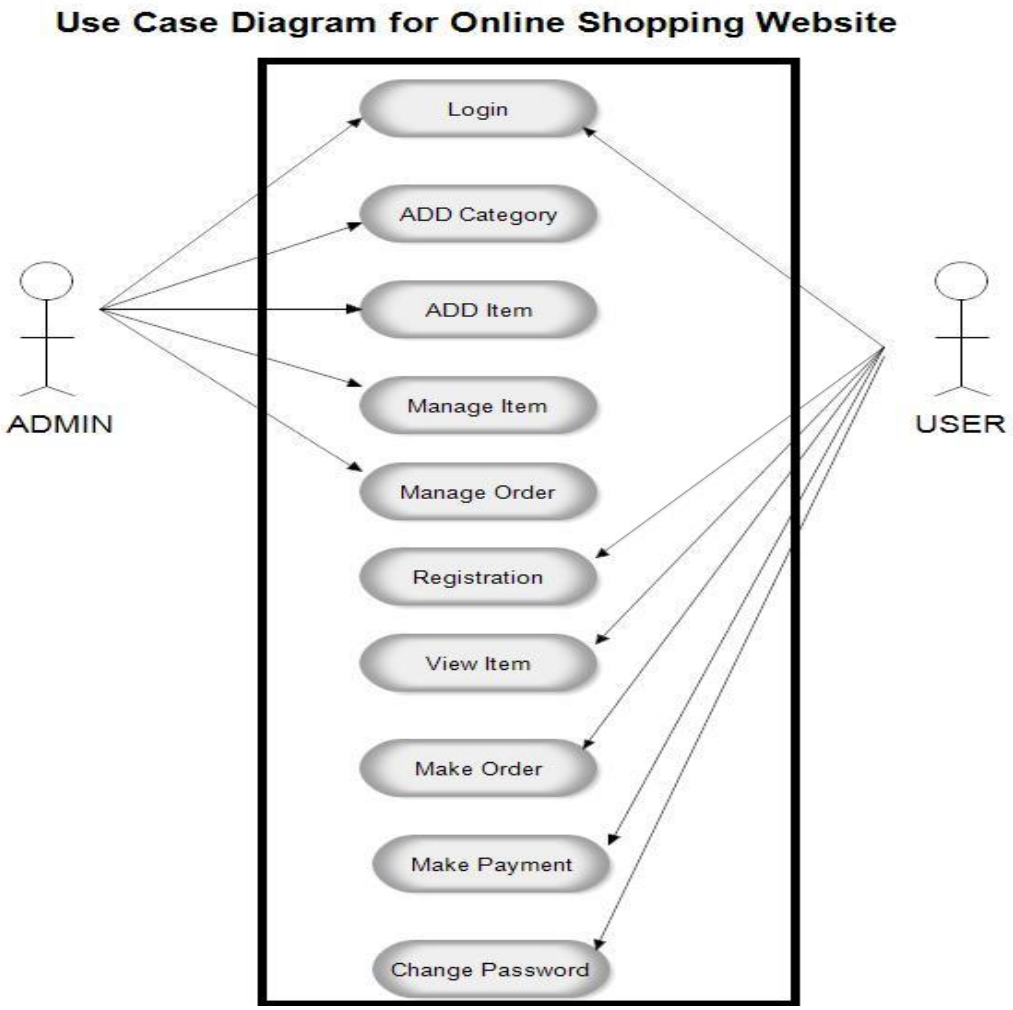



*Figure 3: example of use-case (Muller, 1997, page 127)*

*An example of use case diagram for Bank ATM subsystem*




**Use Case Diagram for Bank ATM System**




Basic Sequence Diagram 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.



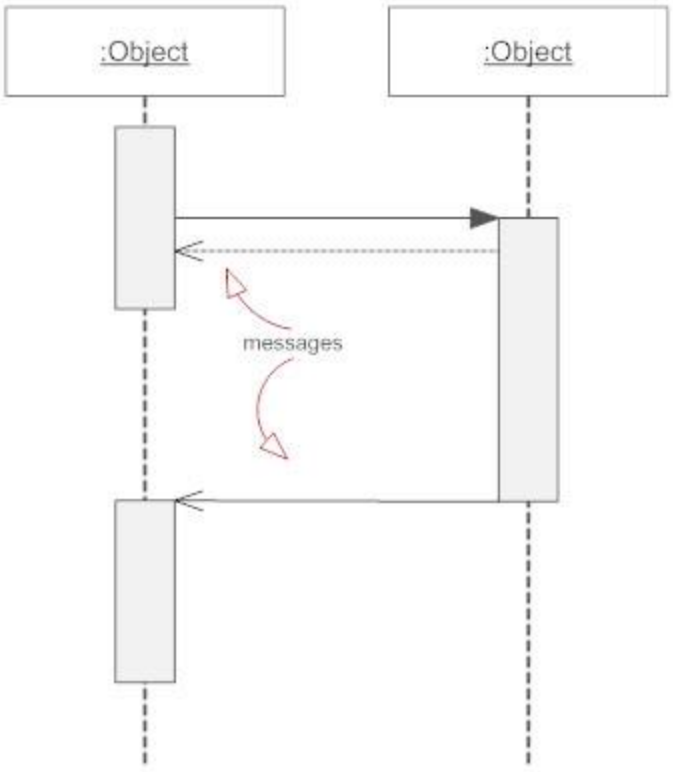

**Activation or Execution Occurrence**
Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.



Activation or Execution Occurrence

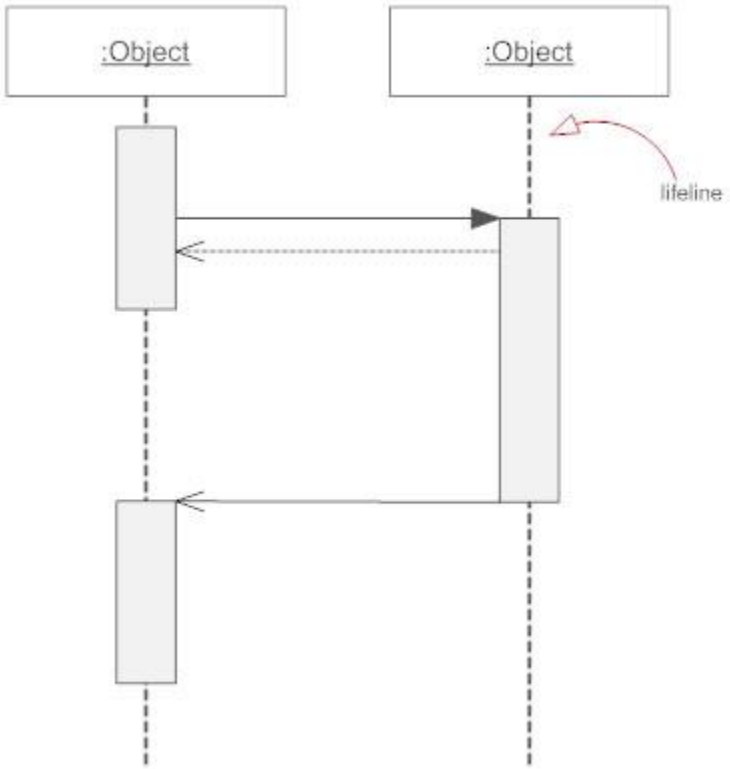
**Messages**

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.



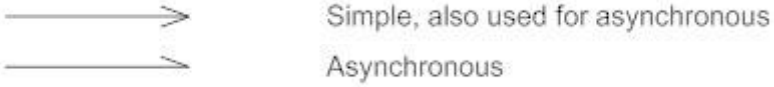

**Destroying Objects**
Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

**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.

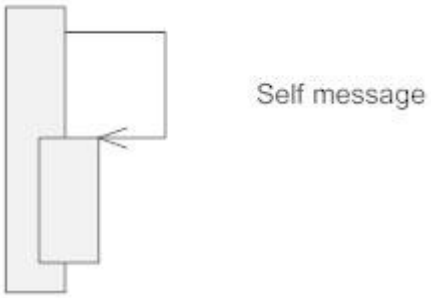


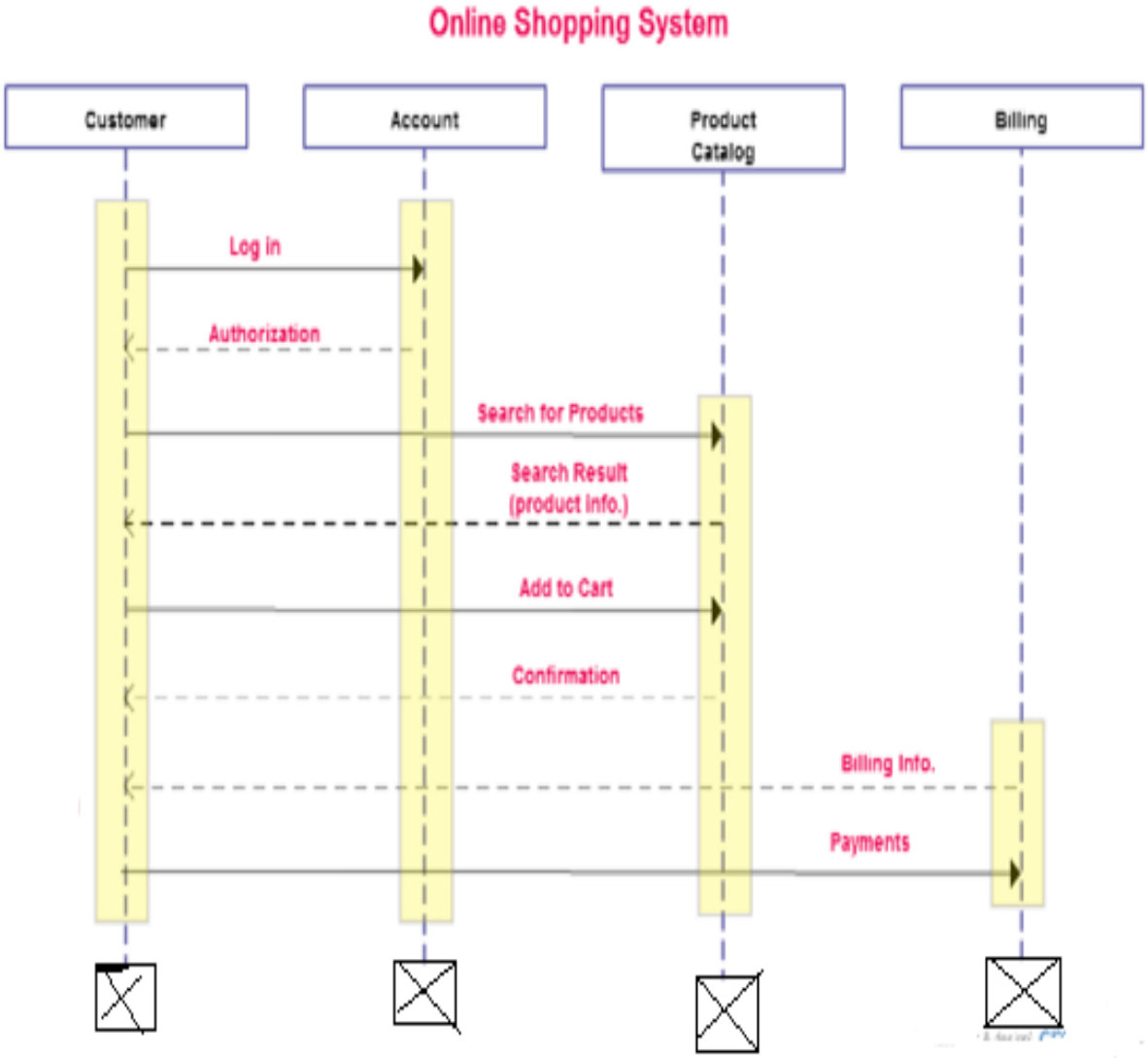

**Reply or Return Message**
A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

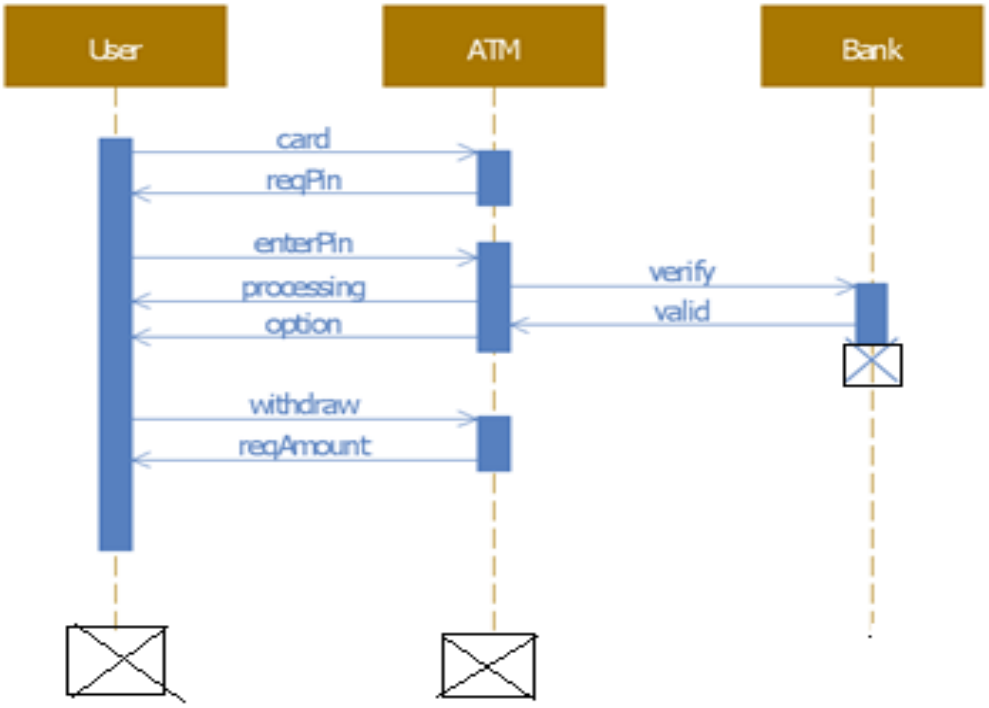


**Self Message**

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






**Difference between a Use case diagram and an Activity diagram**

An activity diagram is used to model the workflow depicting conditions, constraints, sequential and concurrent activities. On the other hand, the purpose of a Use Case is to just depict the functionality i.e. what the system does and not how it is done. So in simple terms, an activity diagram shows 'How' while a Use case shows 'What' for a particular system.

The levels of abstraction also vary for both of them. An activity diagram can be used to illustrate a business process (high level implementation) to a stand alone algorithm (ground level implementation). However, Use cases have a low level of abstraction. They are used to show a **high level** of implementation only.

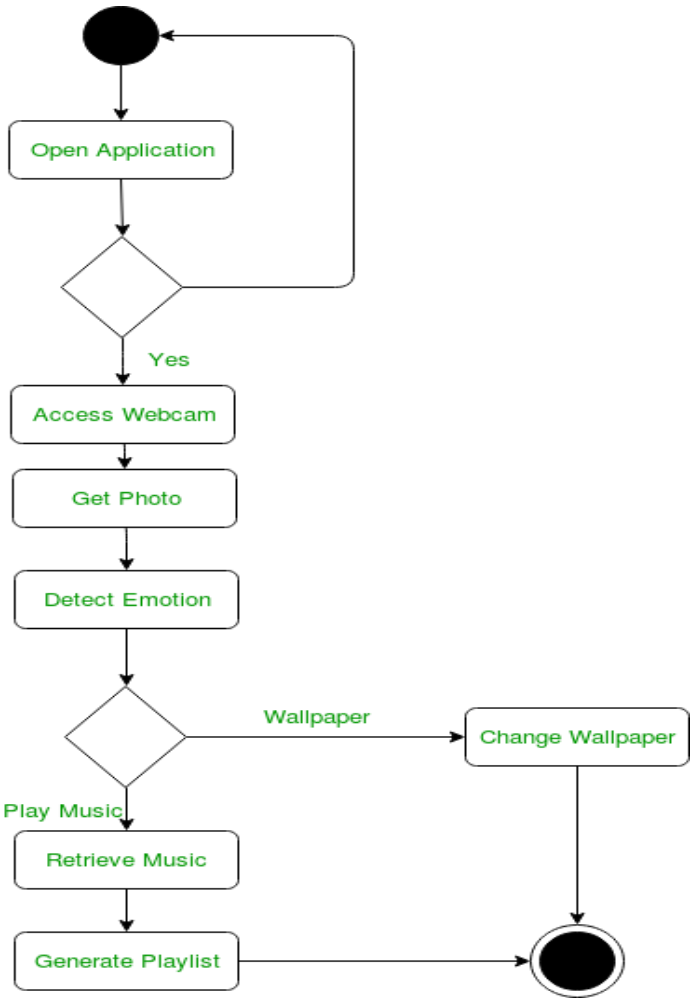



**Figure –** an activity diagram for an emotion based music player
The above figure depicts an activity diagram for an emotion based music player which can also be used to change the wallpaper.

The various components used in the diagram and the standard notations are explained below.

## Activity Diagram Notations –

1. **Initial State –** The starting state before an activity takes place is depicted using the initial state.

   **Figure –** notation for initial state or start state
   A process can have only one initial state unless we are depicting nested activities. We use a black filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.

For example – Here the initial state is the state of the system before the application is opened.

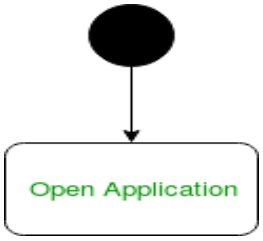



**Figure –** initial state symbol being used

2. **Action or Activity State –** An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.




**Figure –** notation for an activity state

For example – Consider the previous example of opening an application opening the application is an activity state in the activity diagram.

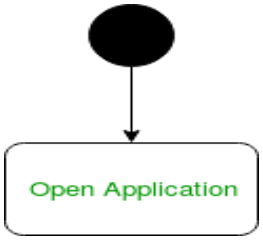



**Figure –** activity state symbol being used

3. **Action Flow or Control flows –** Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another.



**Figure –** notation for control Flow

An activity state can have multiple incoming and outgoing action flows. We use a line with an arrow head to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.

Consider the example – Here both the states transit into one final state using action flow symbols i.e. arrows.

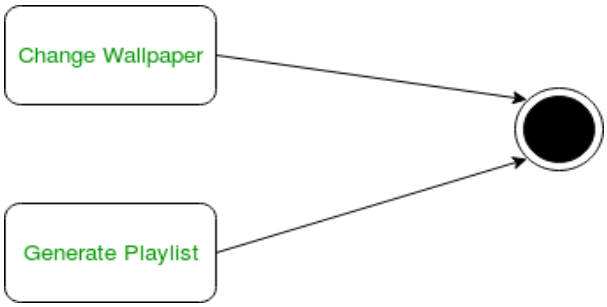



**Figure –** using action flows for transitions

4. **Decision node and Branching –** When we need to make a decision before deciding the flow of control, we use the decision node.

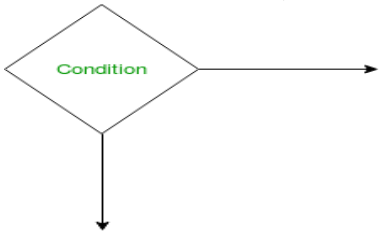



**Figure –** notation for decision node

The outgoing arrows from the decision node can be labelled with conditions or guard expressions.It always includes two or more output arrows.

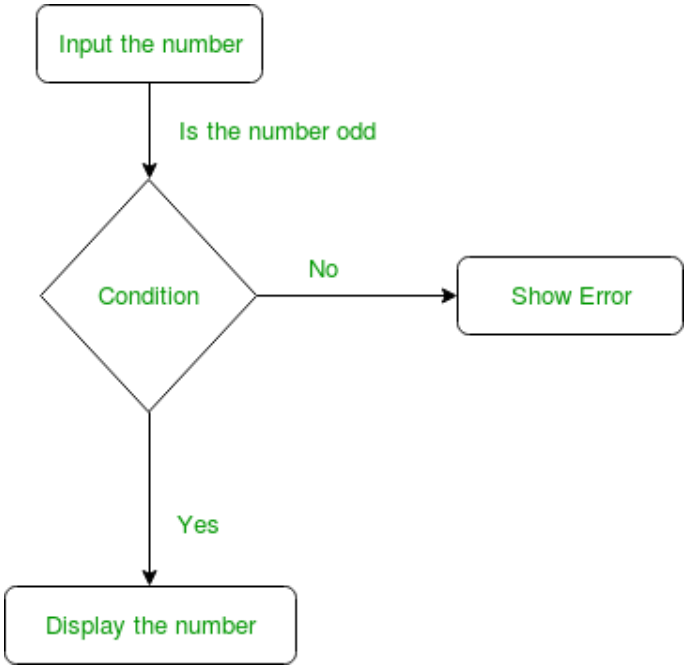



**Figure –** an activity diagram using decision node

.

5. **Fork –** Fork nodes are used to support concurrent activities.

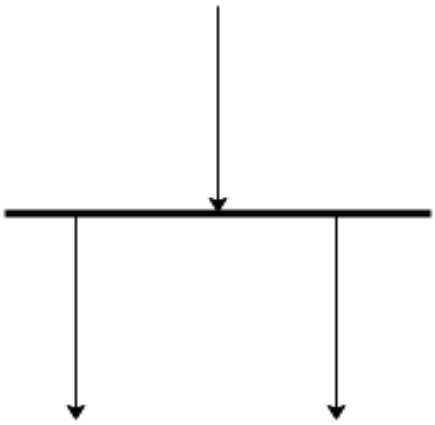


**Figure –** fork notation

When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement.

We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.

For example: In the example below, the activity of making coffee can be split into two concurrent activities and hence we use the fork notation.

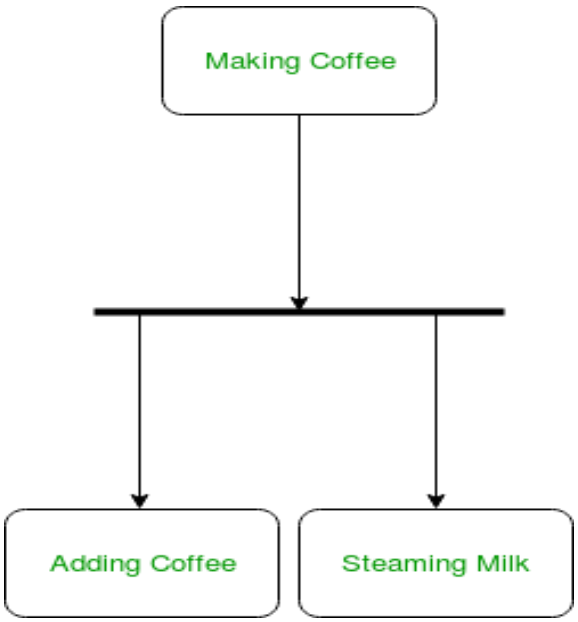



**Figure –** a diagram using fork

6. **Join –** Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.

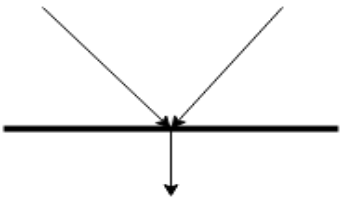


**Figure –** join notation

For example – When both activities i.e. steaming the milk and adding coffee get completed, we converge them into one final activity.

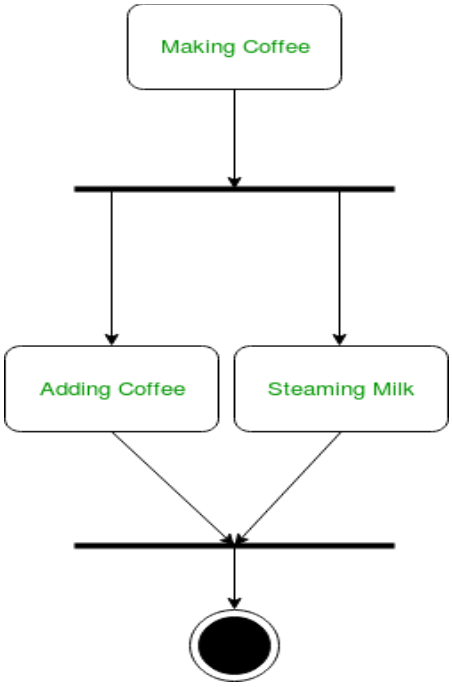



**Figure –** a diagram using join notation

7. **Merge or Merge Event –** Scenarios arise when activities which are not being executed concurrently have to be merged. We use the merge notation for such scenarios. We can merge two or more activities into one if the control proceeds onto the next activity irrespective of the path chosen.

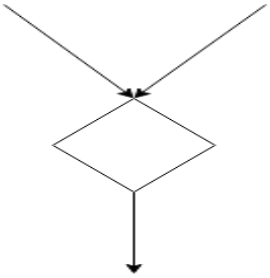


**Figure –** merge notation

For example – In the diagram below: we can't have both sides executing concurrently, but they finally merge into one. A number can't be both odd and even at the same time.

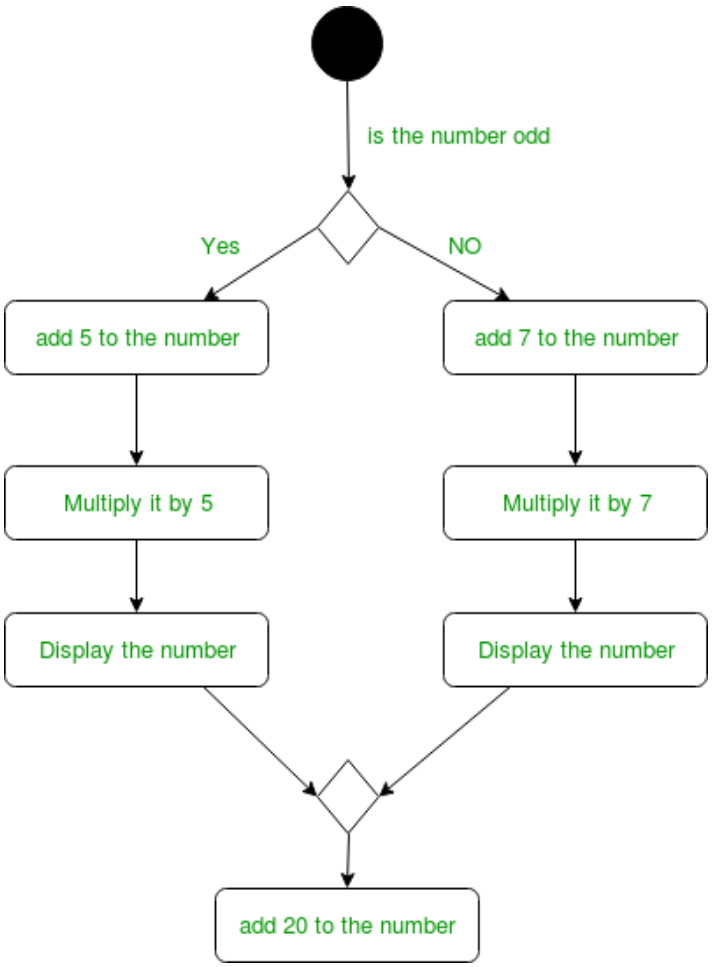



**Figure –** an activity diagram using merge notation

8. **Swimlanes –** We use swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal. Swimlanes are used to add modularity to the activity diagram. It is not mandatory to use swimlanes. They usually give more clarity to the activity diagram. It's similar to creating a function in a program. It's not mandatory to do so, but, it is a recommended practice.

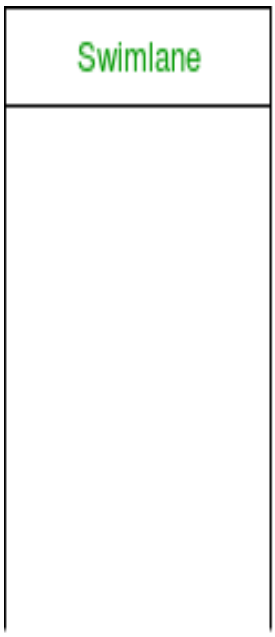



**Figure –** swimlanes notation

We use a rectangular column to represent a swimlane as shown in the figure above. For example – Here different set of activities are executed based on if the number is odd or even. These activities are grouped into a swimlane.

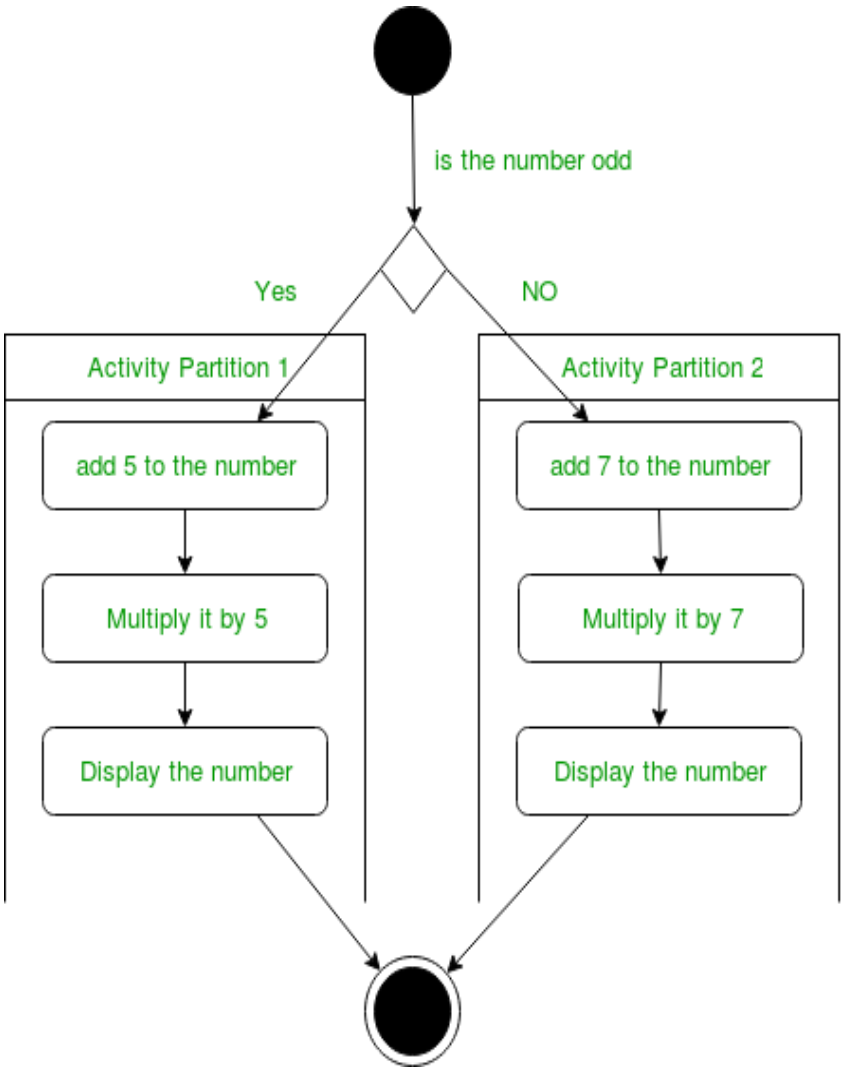



**Figure –** an activity diagram making use of swimlanes
**Figure –** an activity diagram using time event

9. **Final State or End State –** The state which the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.



**Figure –** notation for final state

## How to Draw an activity diagram –

1. Identify the initial state and the final states.
2. Identify the intermediate activities needed to reach the final state from the initial state.
3. Identify the conditions or constraints which cause the system to change control flow.
4. Draw the diagram with appropriate notations.

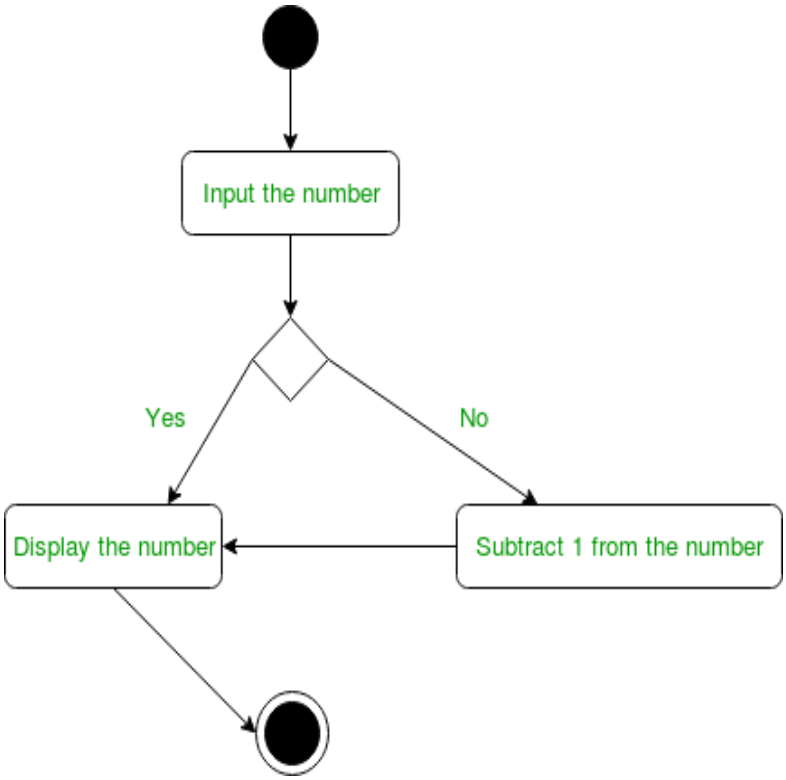
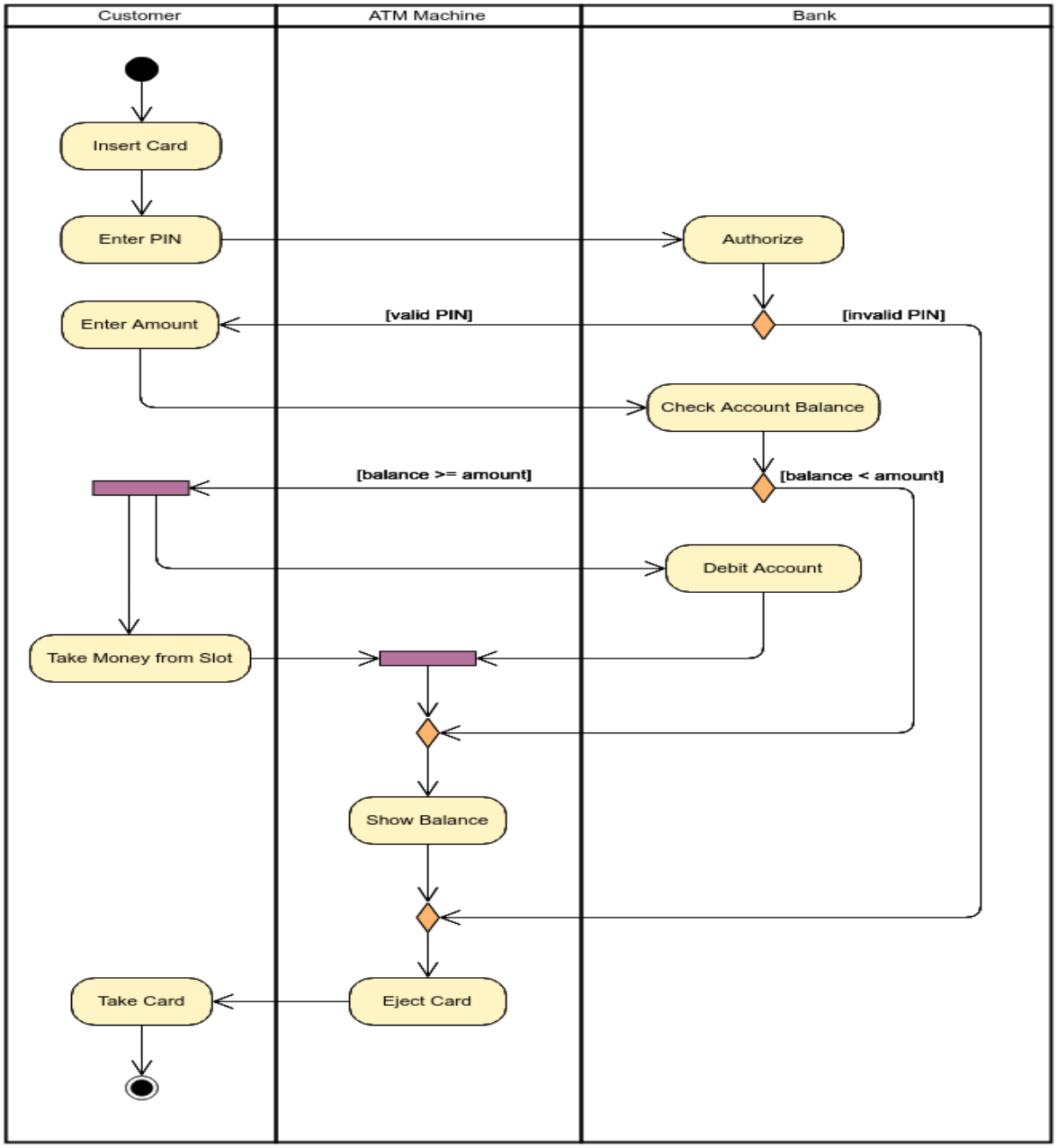


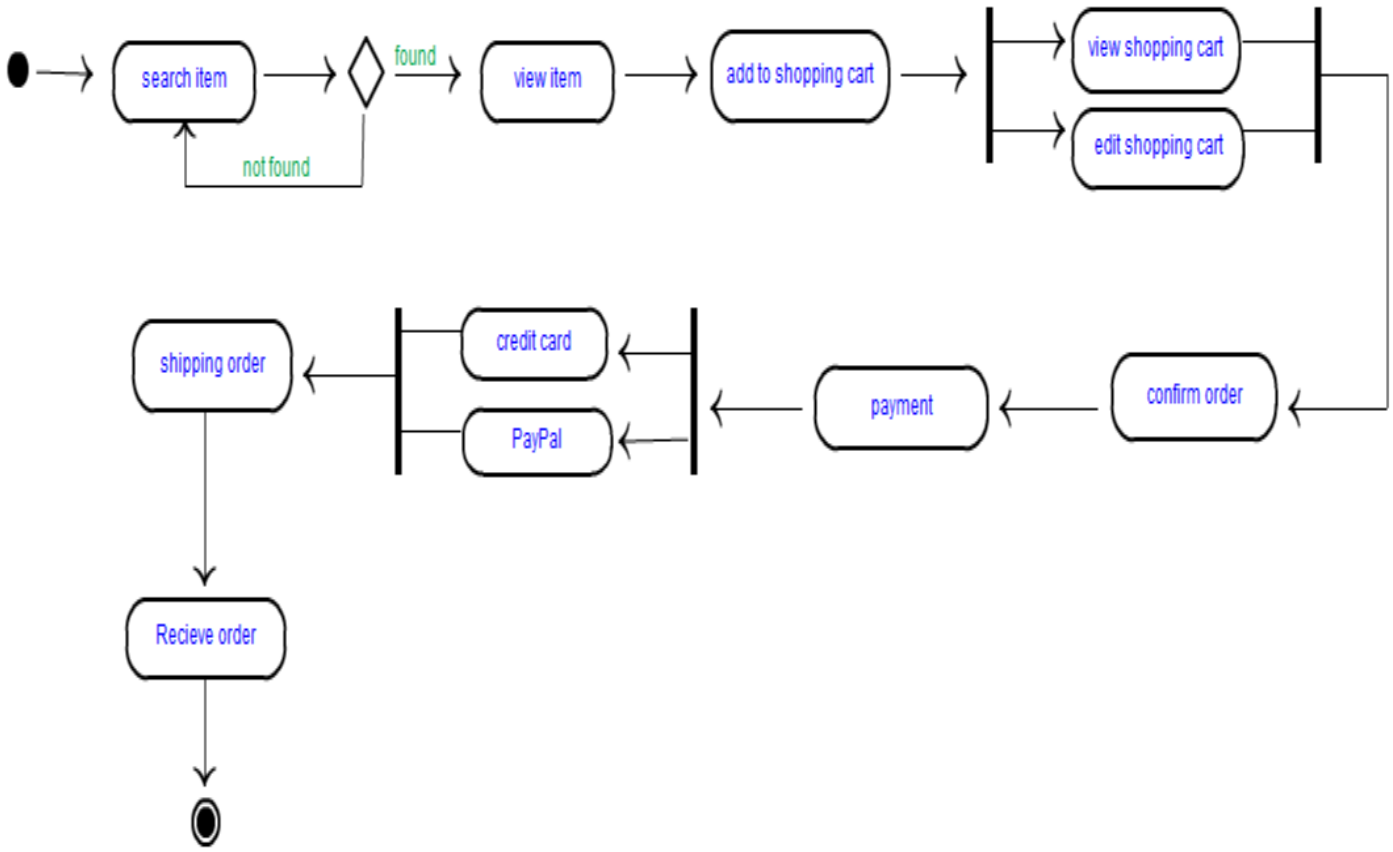

**Figure –** an activity diagram

The above diagram prints the number if it is odd otherwise it subtracts one from the number and displays it.

## Uses of an Activity Diagram –

- Dynamic modelling of the system or a process.
- Illustrate the various steps involved in a UML use case.
- Model software elements like methods,operations and functions.
- We can use Activity diagrams to depict concurrent activities easily.
- Show the constraints, conditions and logic behind algorithms.


Customer
ATM Machine
Bank
Insert Card
Enter PIN
Authorize
Enter Amount
[valid PIN]
[invalid PIN]
Check Account Balance
[balance >= amount]
[balance < amount]
Debit Account
Take Money from Slot
Show Balance
Take Card
Eject Card




**Difference between state diagram and flowchart –**
The basic purpose of a **state diagram** is to portray various changes in state of the class and not the processes or commands causing the changes. However, a **flowchart** on the other hand portrays the processes or commands that on execution change the state of class or an object of the class.

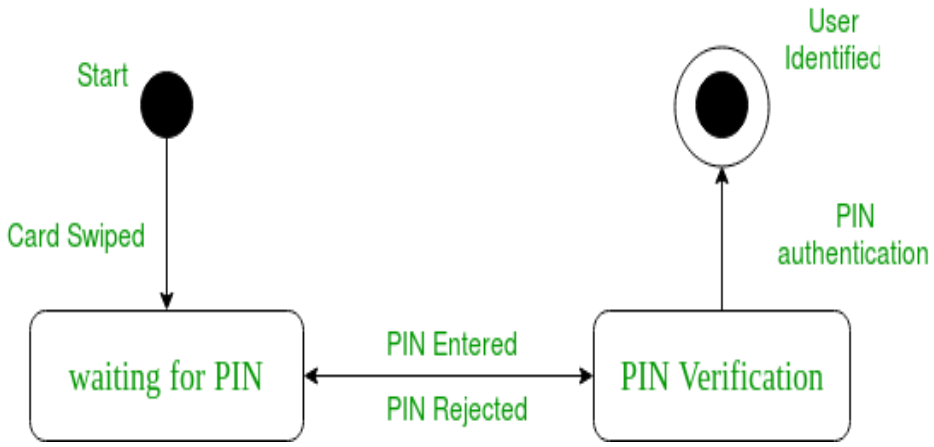



**Figure –** a state diagram for user verification

The state diagram above shows the different states in which the verification sub-system or class exist for a particular system.

## Basic components of a statechart diagram –

1. **Initial state –** We use a black filled circle represent the initial state of a System or a class.



**Figure –** initial state notation

2. **Transition –** We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.




**Figure –** transition

3. **State –** We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.




**Figure –** state notation

4. **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. We use the fork notation to represent a state splitting into two or more concurrent states.

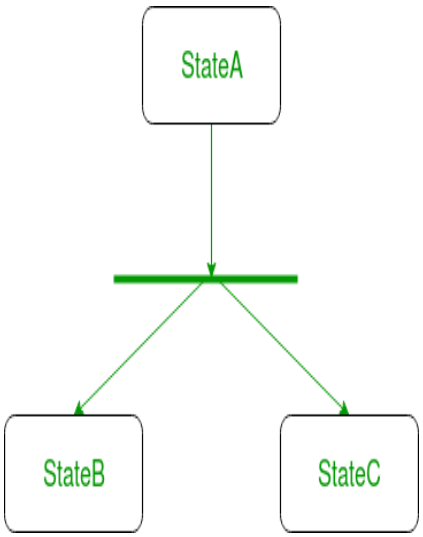



**Figure –** a diagram using the fork notation

5. **Join –** We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state. We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.

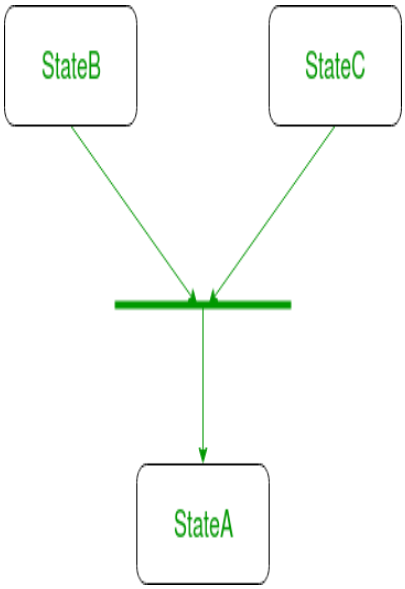



**Figure –** a diagram using join notation

6. **Self transition –** We use a solid arrow pointing back to the state itself to represent a self transition. There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self transitions to represent such cases.

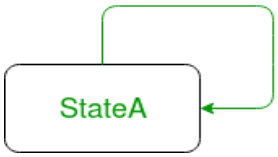



**Figure –** self transition notation

7. **Final state –** We use a filled circle within a circle notation to represent the final state in a state machine diagram.



**Figure –** final state notation

## Steps to draw a state diagram –

1. Identify the initial state and the final terminating states.
2. Identify the possible states in which the object can exist (boundary values corresponding to different attributes guide us in identifying different states).
3. Label the events which trigger these transitions.

**Example –** state diagram for an online order –

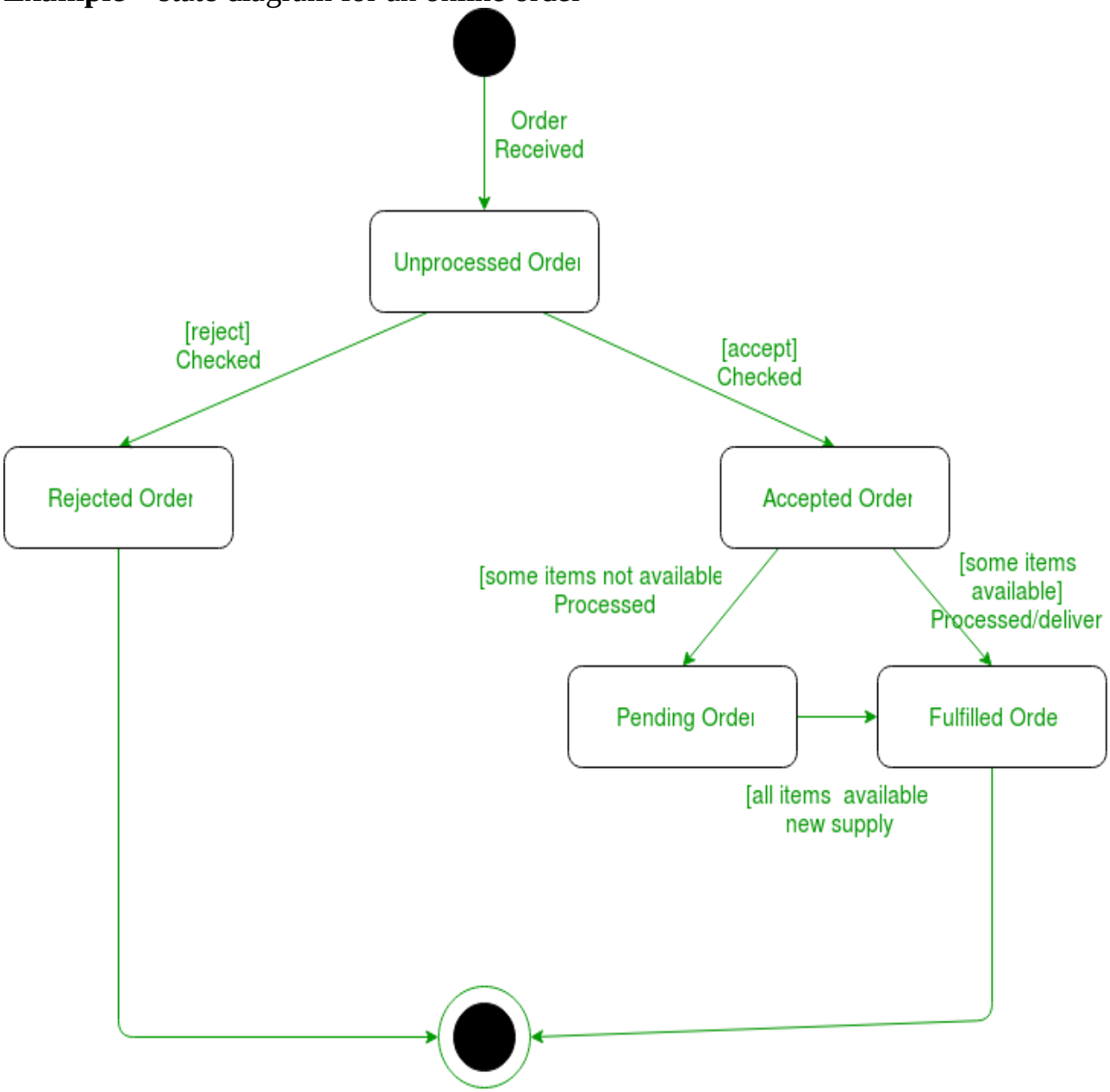
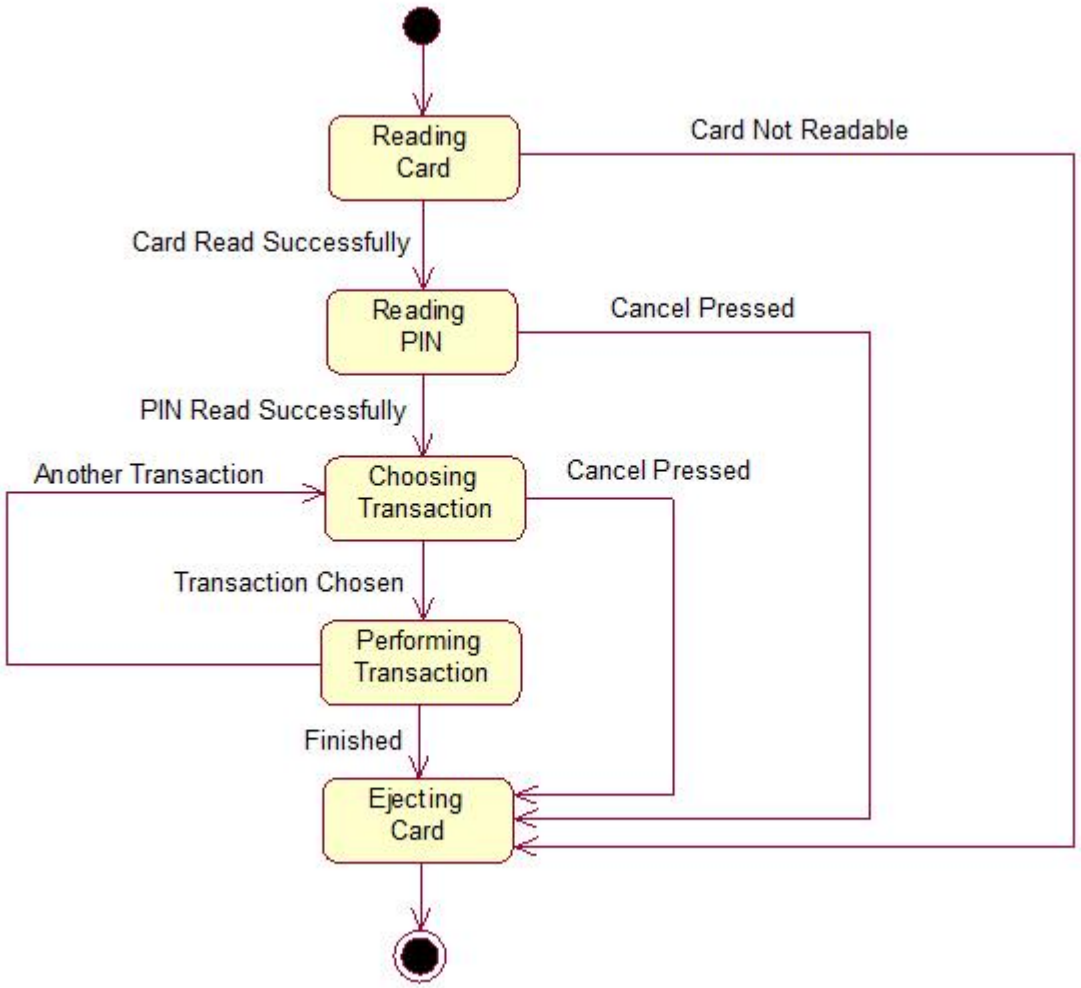


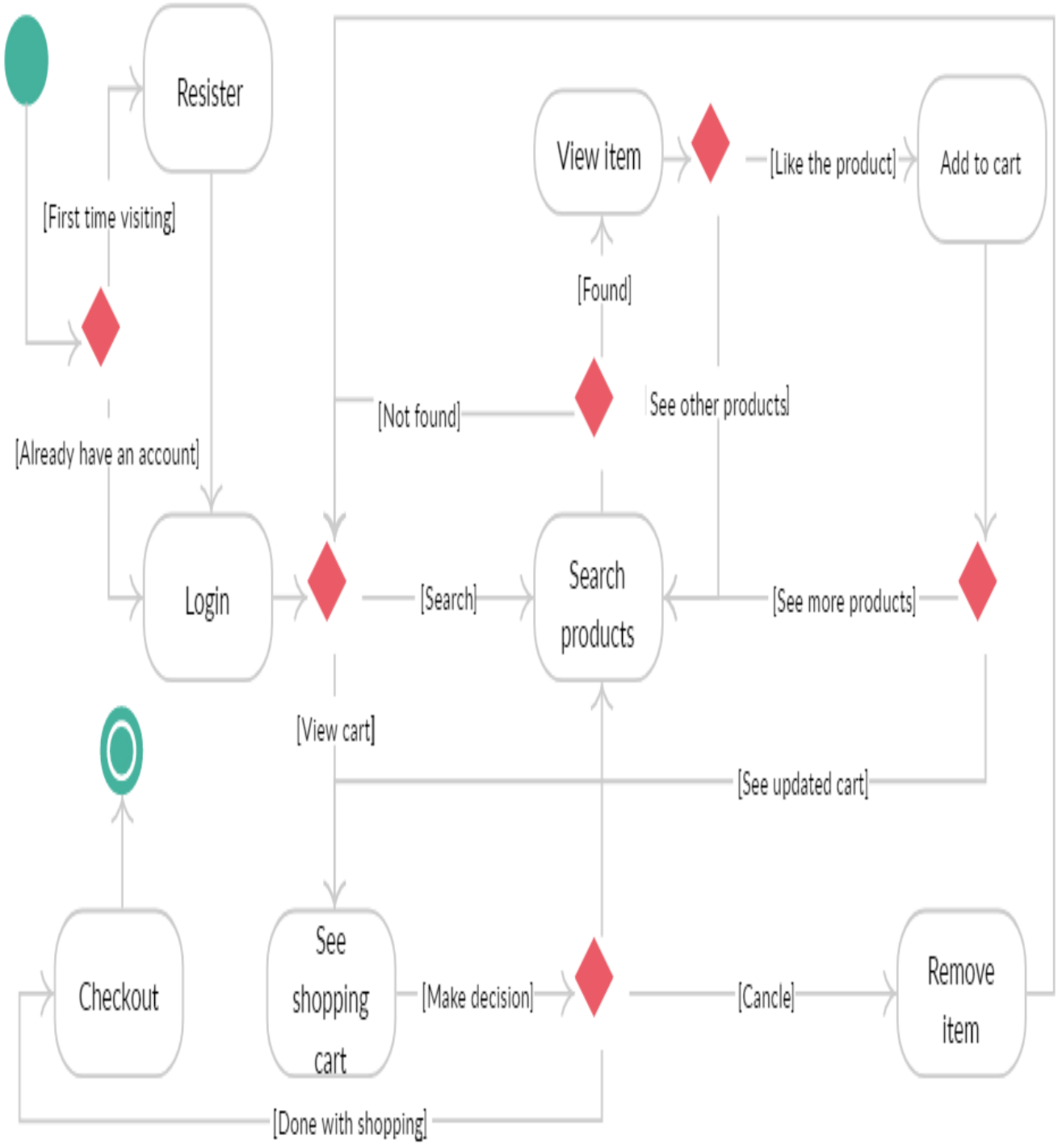
**Figure –** state diagram for an online order

The UMl diagrams we draw depend on the system we aim to represent. Here is just an example of how an online ordering system might look like :

1. On the event of an order being received, we transit from our initial state to Unprocessed order state.
2. The unprocessed order is then checked.
3. If the order is rejected, we transit to the Rejected Order state.
4. If the order is accepted and we have the items available we transit to the fulfilled order state.
5. However if the items are not available we transit to the Pending Order state.
6. After the order is fulfilled, we transit to the final state. In this example, we merge the two states i.e. Fulfilled order and Rejected order into one final state.

**Note –** Here we could have also treated fulfilled order and rejected order as final states separately.




Resister
[First time visiting]
[Already have an account]
Login
[Search]
Search products
View item
[Like the product]
Add to cart
[Found]
[Not found]
See other products
[See more products]
[View cart]
[See updated cart]
Checkout
See shopping cart
[Make decision]
[Cancle]
Remove item
[Done with shopping]

**They 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. • Collaboration diagrams show (used to model) how objects interact and their roles. • They are very similar to sequence diagrams. Actually they are considered as a cross between class and sequence diagram. • Sequence Diagrams are arranged according to Time. • Collaboration Diagrams represent the structural organization of object. • [Both sequence and collaboration diagrams are called interaction diagrams] • Forms a context for interactions o May realize use cases o May be associated with operations o May describe the static structure of classes**

Notations of Collaboration Diagram

**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:

- Each object in the collaboration is named and has its class specified
- Not all classes need to appear
- There may be more than one object of a class
- 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.

**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.

- Each Actor is named and has a role
- One actor will be the initiator of the use case

**Links**

Links connect objects and actors and are instances of associations and each link corresponds to an association in the class diagram

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**

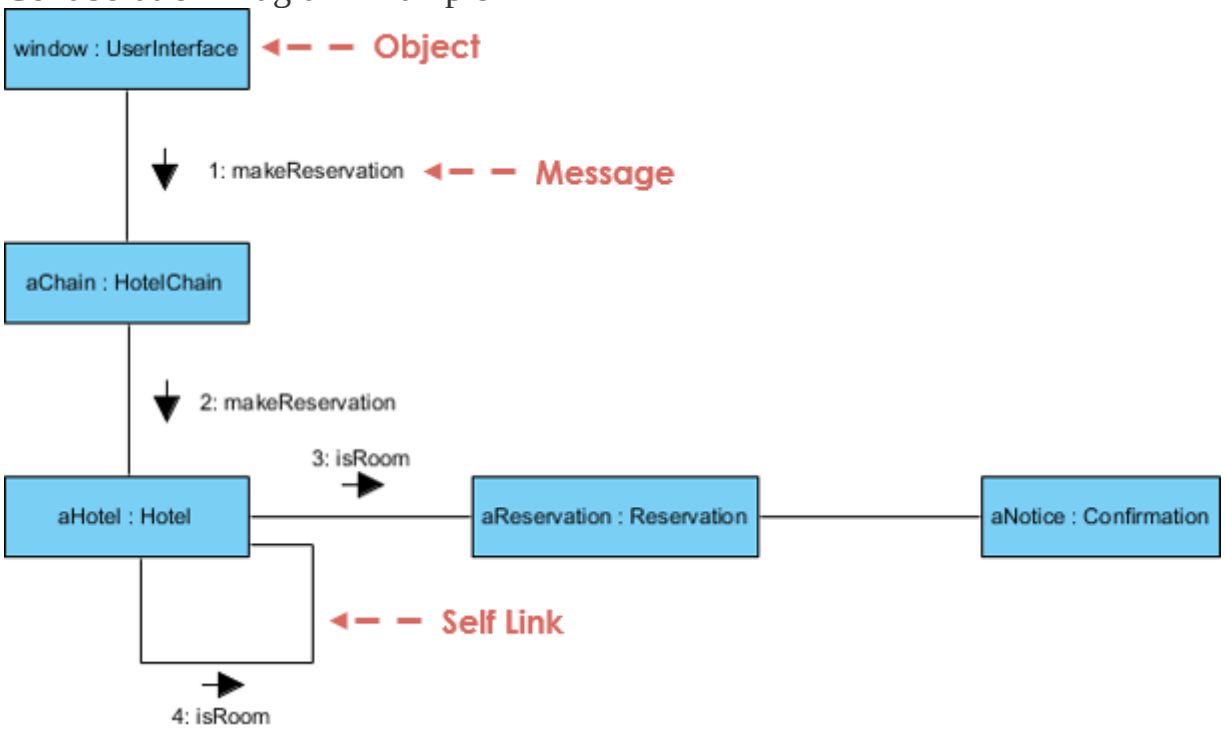
A message is a communication between objects that conveys information with the expectation that activity will ensue. In collaboration diagrams, a message is shown as a labeled arrow placed near a link.

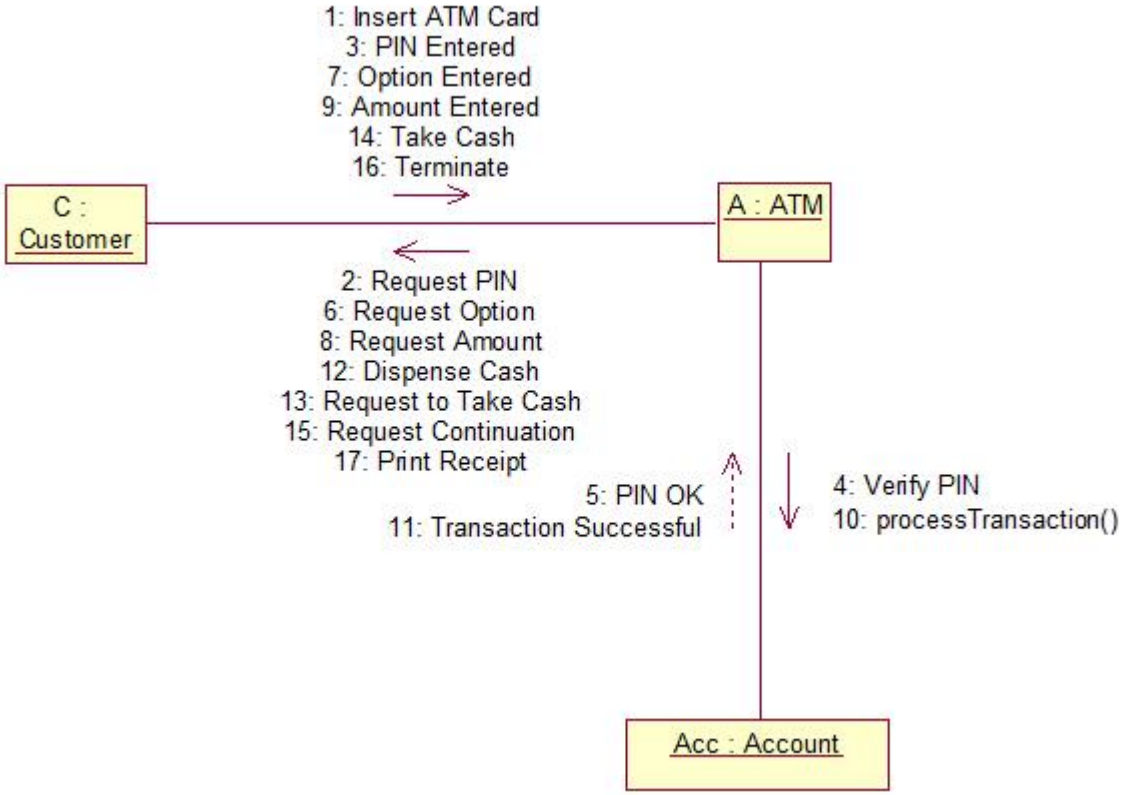
- The message is directed from sender to receiver
- The receiver must understand the message
- The association must be navigable in that direction

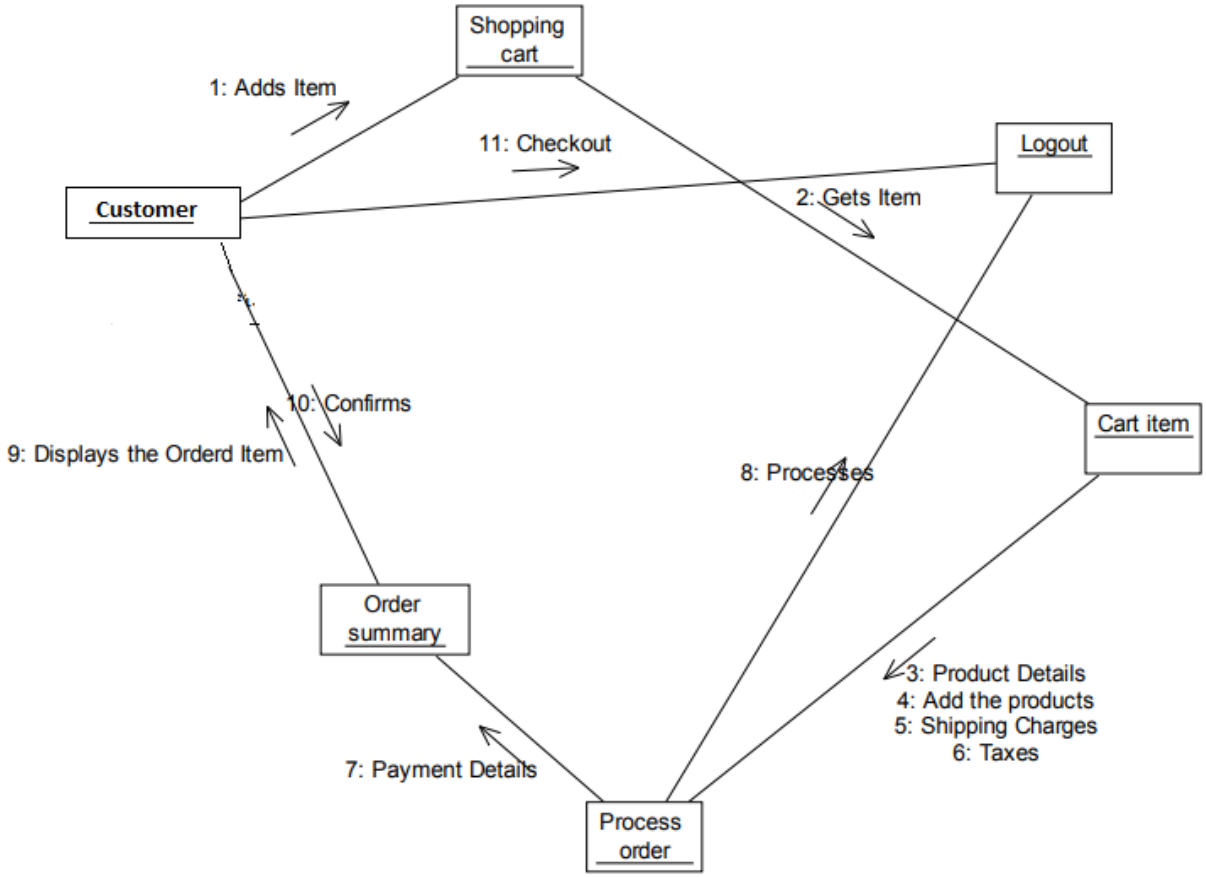
Steps for Creating Collaboration Diagrams

1. Identify behavior whose realization and implementation is specified
2. Identify the structural elements (class roles, objects, subsystems) necessary to carry out the functionality of the collaboration
    - Decide on the context of interaction: system, subsystem, use case and operation
3. Model structural relationships between those elements to produce a diagram showing the context of the interaction
4. Consider the alternative scenarios that may be required
    - Draw instance level collaboration diagrams, if required.
    - Optionally draw a specification level collaboration diagram to summarize the alternative scenarios in the instance level sequence diagrams

Collaboration Diagram Example





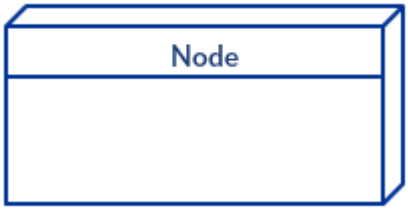

Deployment Diagram :

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed. A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them.

## Deployment Diagram Notations

In order to draw a deployment diagram, you need to first become familiar with the following deployment diagram notations and deployment diagram elements.

***Nodes***



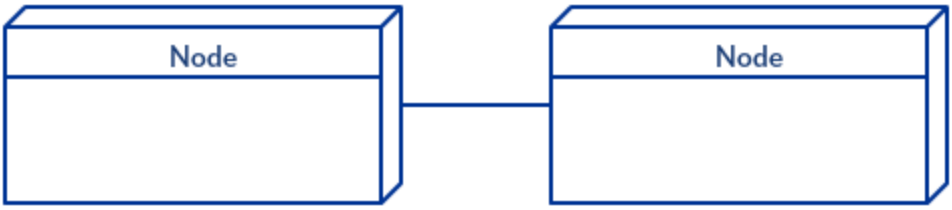

A node, represented as a cube, is a physical entity that executes one or more components, subsystems or executables. A node could be a hardware or software element.

***Artifacts***



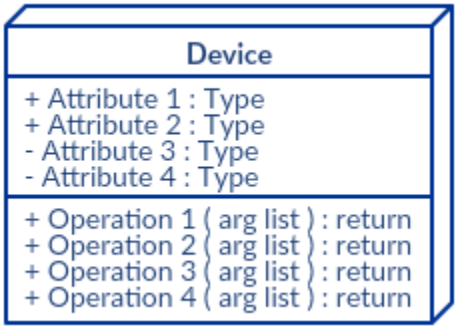

Artifacts are concrete elements that are caused by a development process. Examples of artifacts are libraries, archives, configuration files, executable files etc.

***Communication Association***



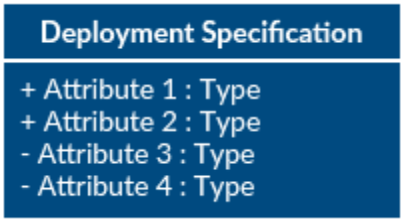

This is represented by a solid line between two nodes. It shows the path of communication between nodes.

***Devices***



A device is a node that is used to represent a physical computational resource in a system. An example of a device is an application server.

***Deployment Specifications***




Deployment specifications is a configuration file, such as a text file or an XML document. It describes how an artifact is deployed on a node.

**How to Draw a Deployment Diagram**

Follow the simple steps below to draw a deployment diagram. You can either use the deployment diagram examples below to get a head start or use our UML diagram tool to start from the beginning.

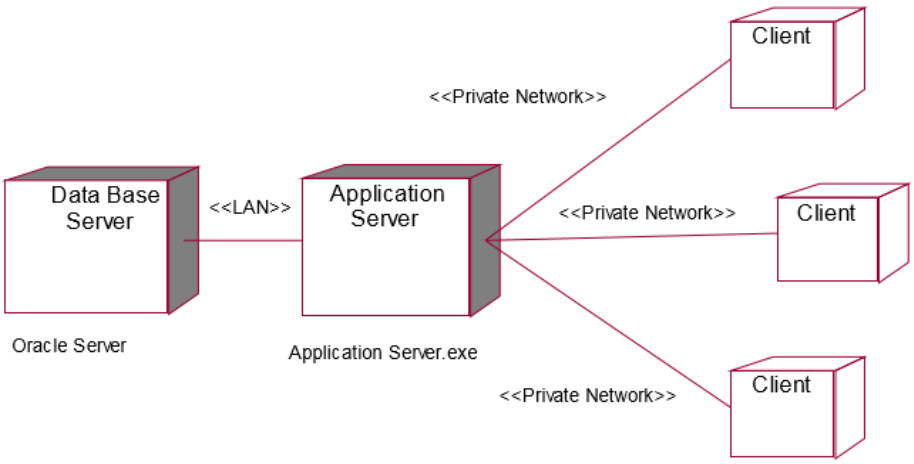
**Step 1:** Identify the purpose of your deployment diagram. And to do so, you need to identify the nodes and devices within the system you'll be visualizing with the diagram.

**Step 2:** Figure out the relationships between the nodes and devices. Once you know how they are connected, proceed to add the communication associations to the diagram.

**Step 3:** Identify what other elements like components, active objects you need to add to complete the diagram.

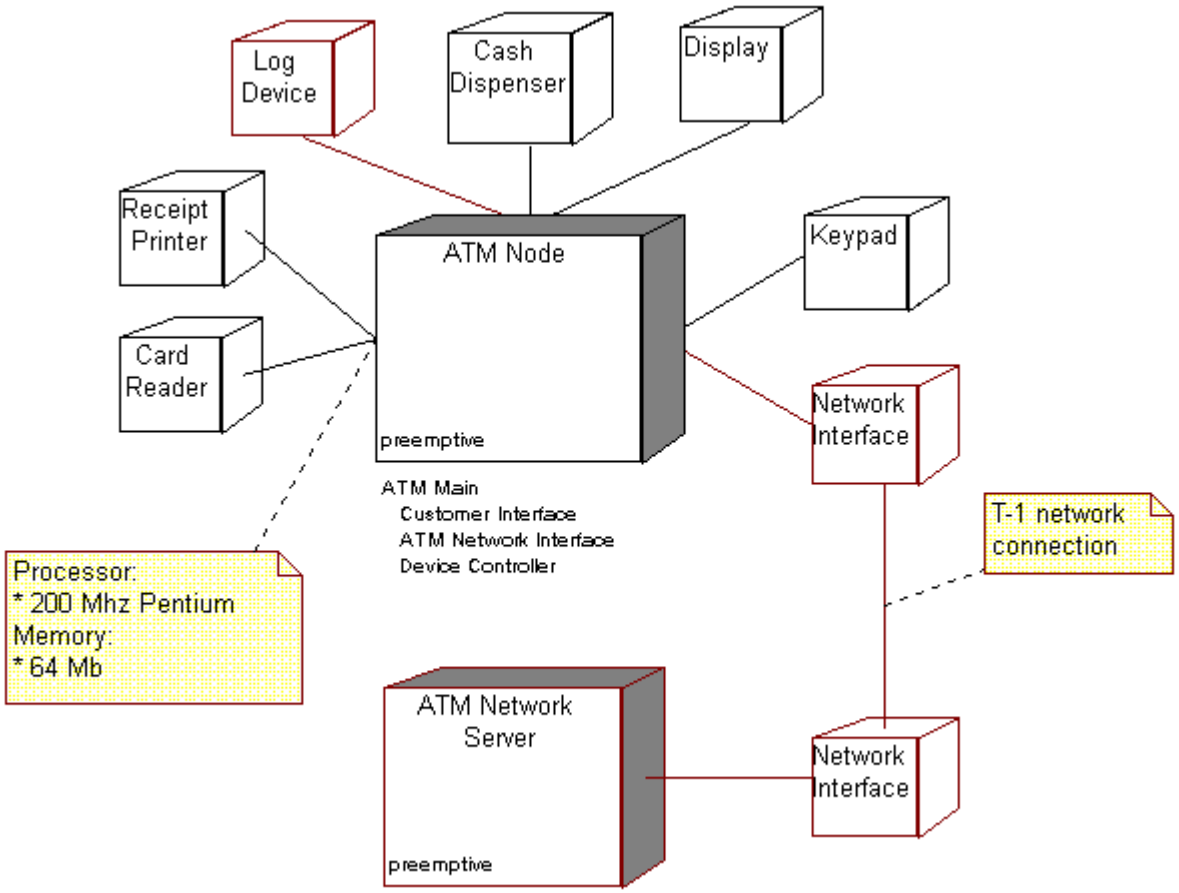
**Step 4:** Add dependencies between components and objects as required.

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

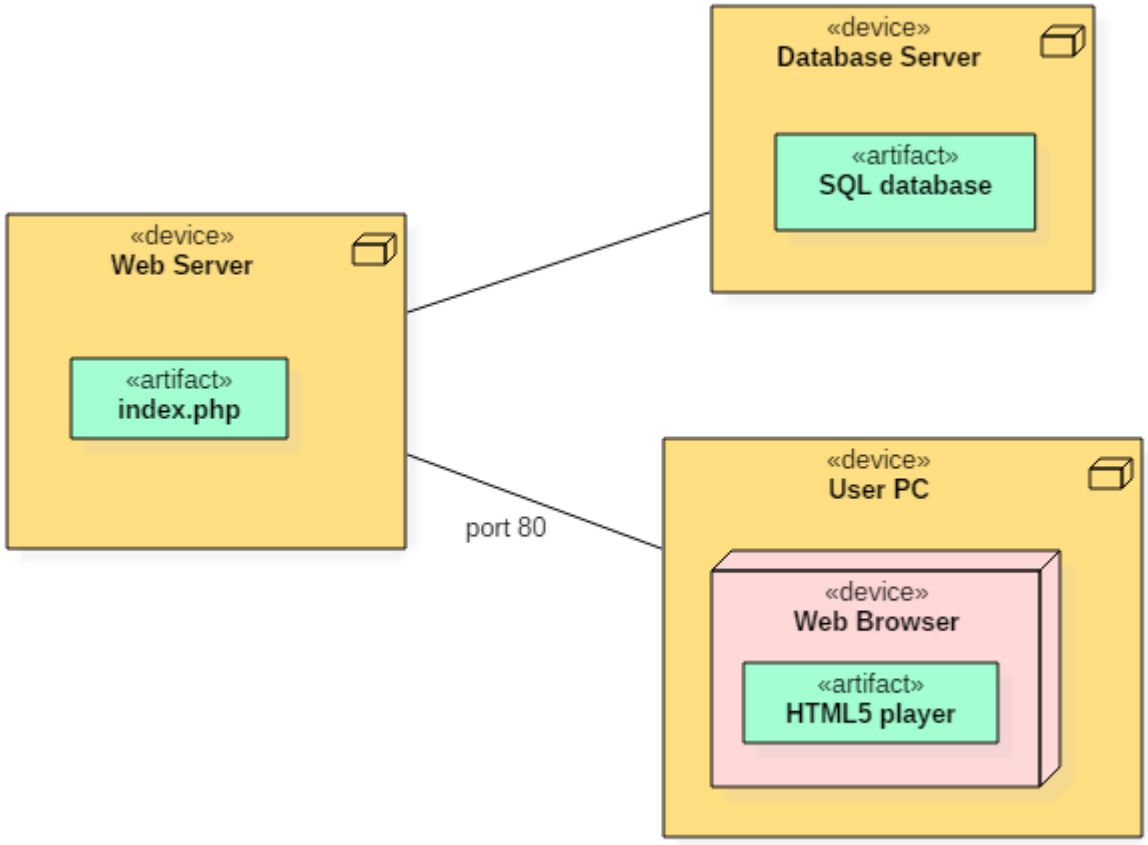



.

Fig 9..2 Deployment diagram for Online Shopping System.



Following deployment diagram represents the working of HTML5 video player in the browser:




Deployment Diagram

**A component diagram**, also known as a UML component diagram, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.

**What is a Component?**

A component is a replaceable and executable piece of a system whose implementation details are hidden. A component provides the set of interfaces that a component realizes or implements. Components also require interfaces to carry out a function.

Basic Component Diagram Symbols and Notations

***Component***

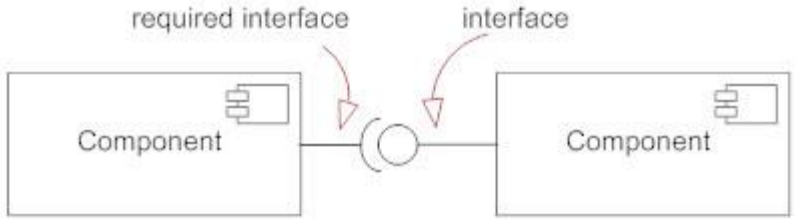
A component is a logical unit block of the system, a slightly higher abstraction than classes. It is represented as a rectangle with a smaller rectangle in the upper right corner

with tabs or the word written above the name of the component to help distinguish it from a class.




***Interface***

An interface (small circle or semi-circle on a stick) describes a group of operations used (required) or created (provided) by components. A full circle represents an interface created or provided by the component. A semi-circle represents a required interface, like a person's input.



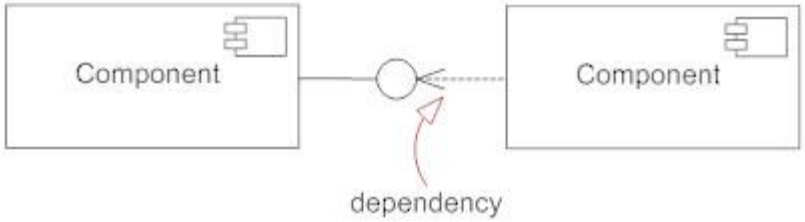

There are two types of interfaces,

1. Provided interfaces
2. Required interfaces

We can connect provided and required interfaces using assembly connector.

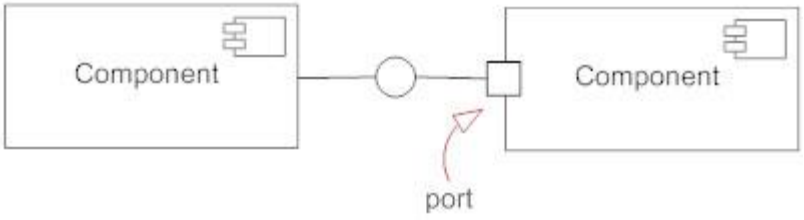
***Dependencies***

Draw dependencies among components using dashed arrows.



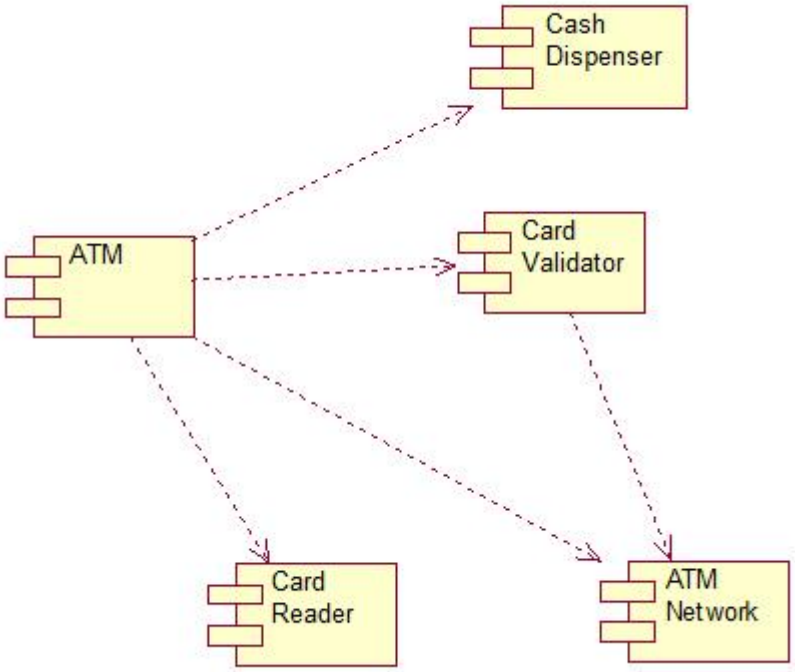

***Port***

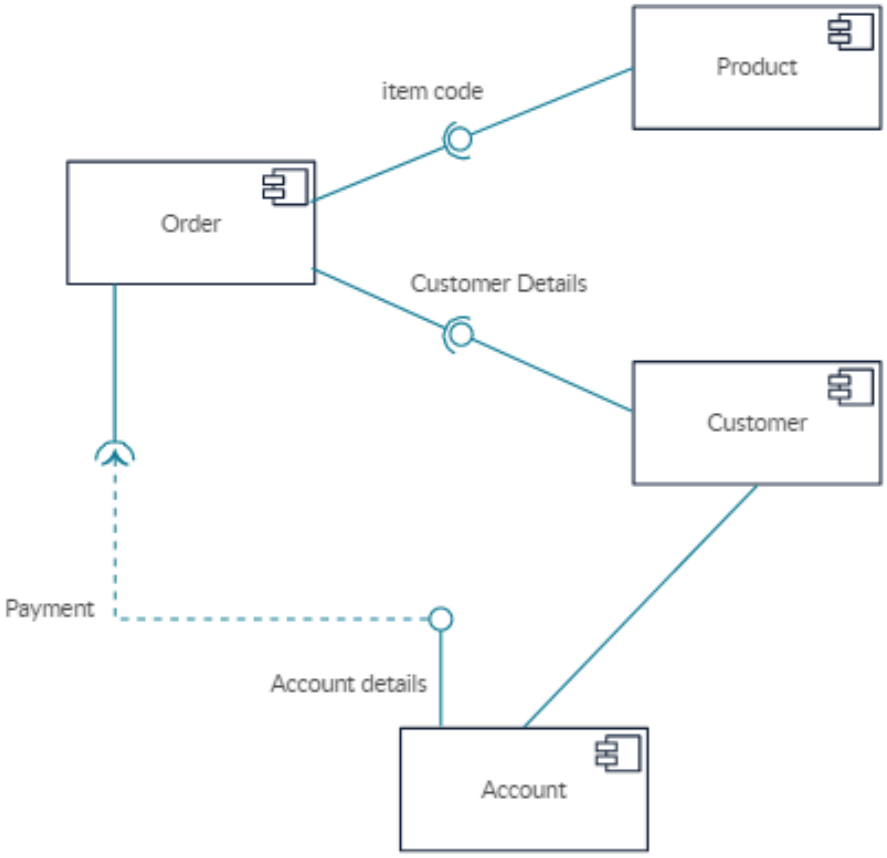
Ports are represented using a square along the edge of the system or a component. A port is often used to help expose required and provided interfaces of a component.



How to Draw a Component Diagram

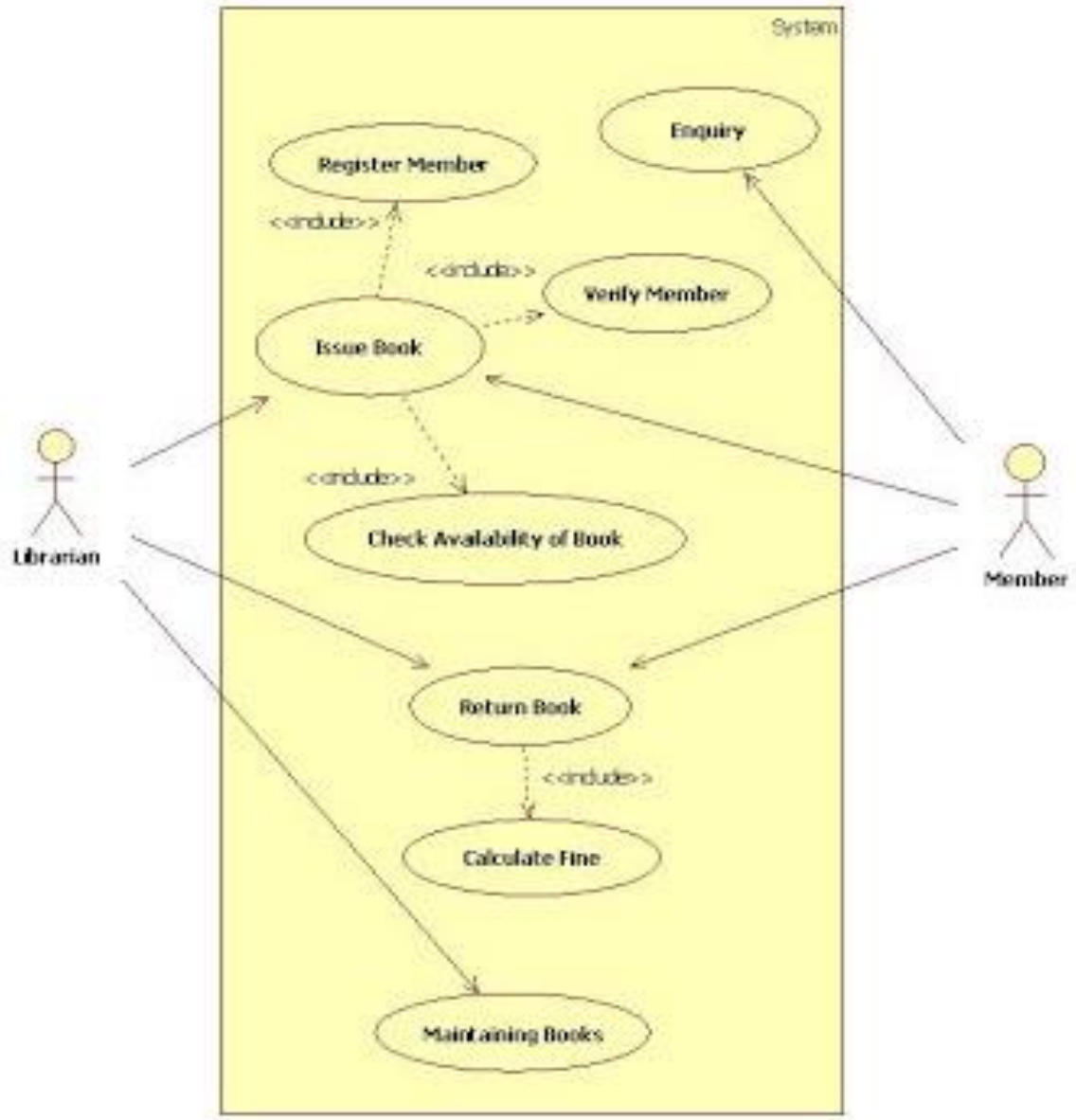
- Take stock of everything needed to implement the planned system. For example, for a simple e-commerce system, you'll need components that describe products, orders, and customer accounts.
- Create a visual for each of the components.
- Describe the organization and relationships between components using interfaces, ports, and dependencies.



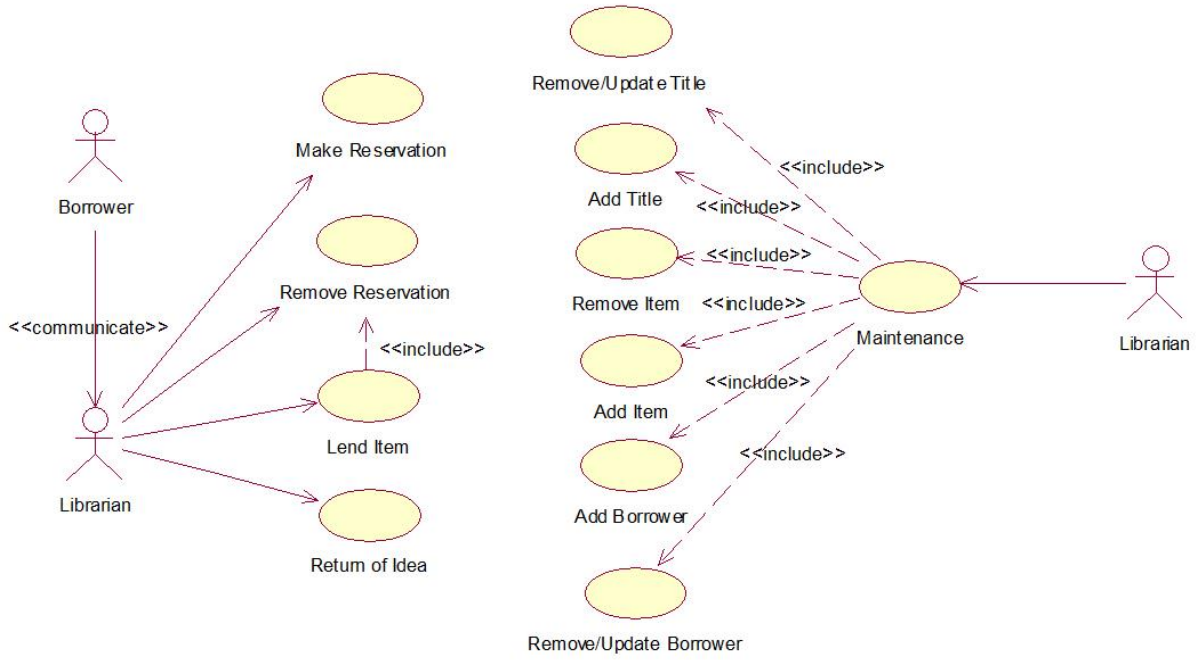



**Online shopping system**

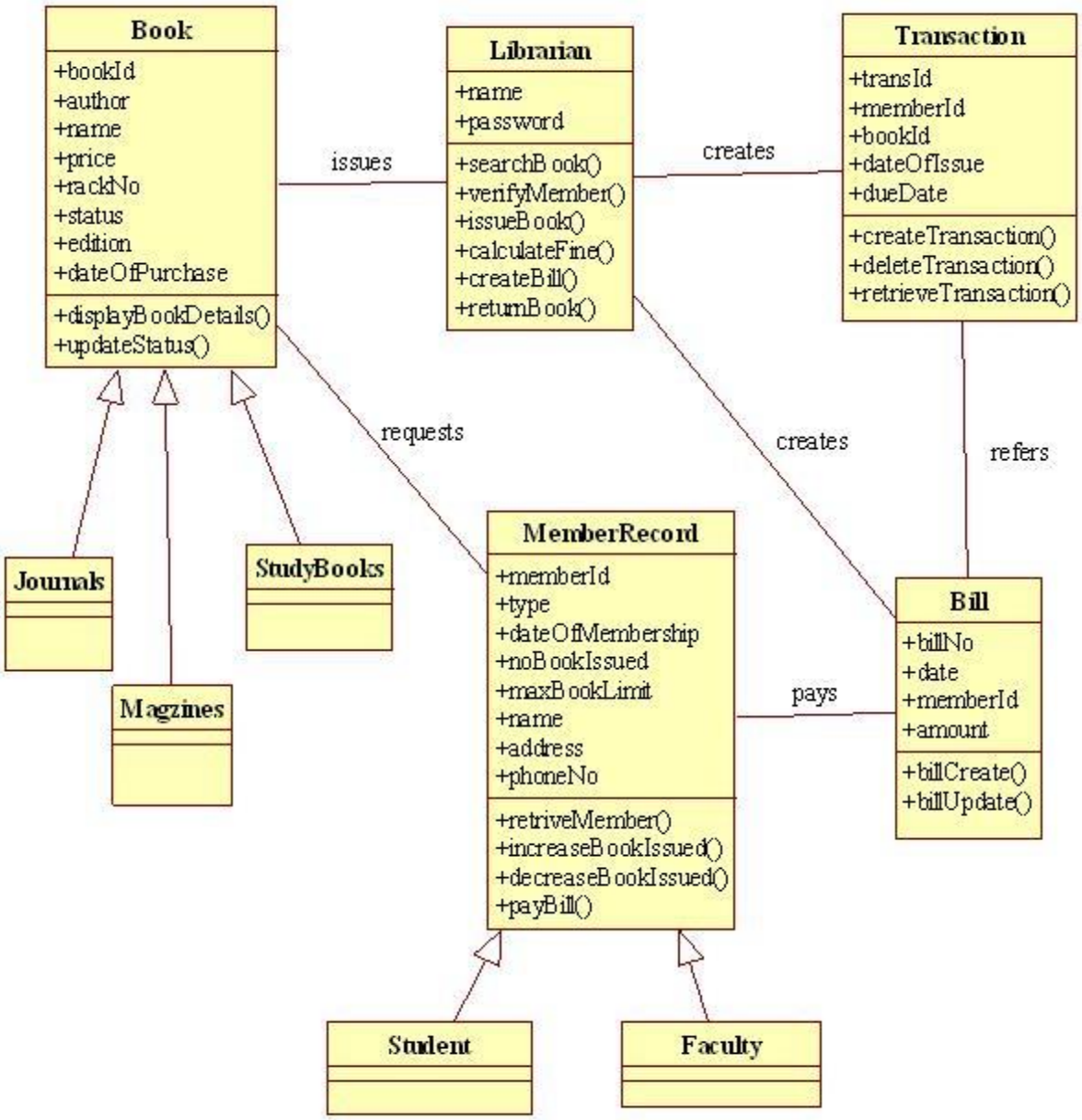
- **<u>Use Case Diagram:- Library Management System</u>**


System
Enquiry
Register Member
<<include>>
<<include>>
Verify Member
Issue Book
<<include>>
Check Availability of Book
Librarian
Member
Return Book
<<include>>
Calculate Fine
Maintaining Books

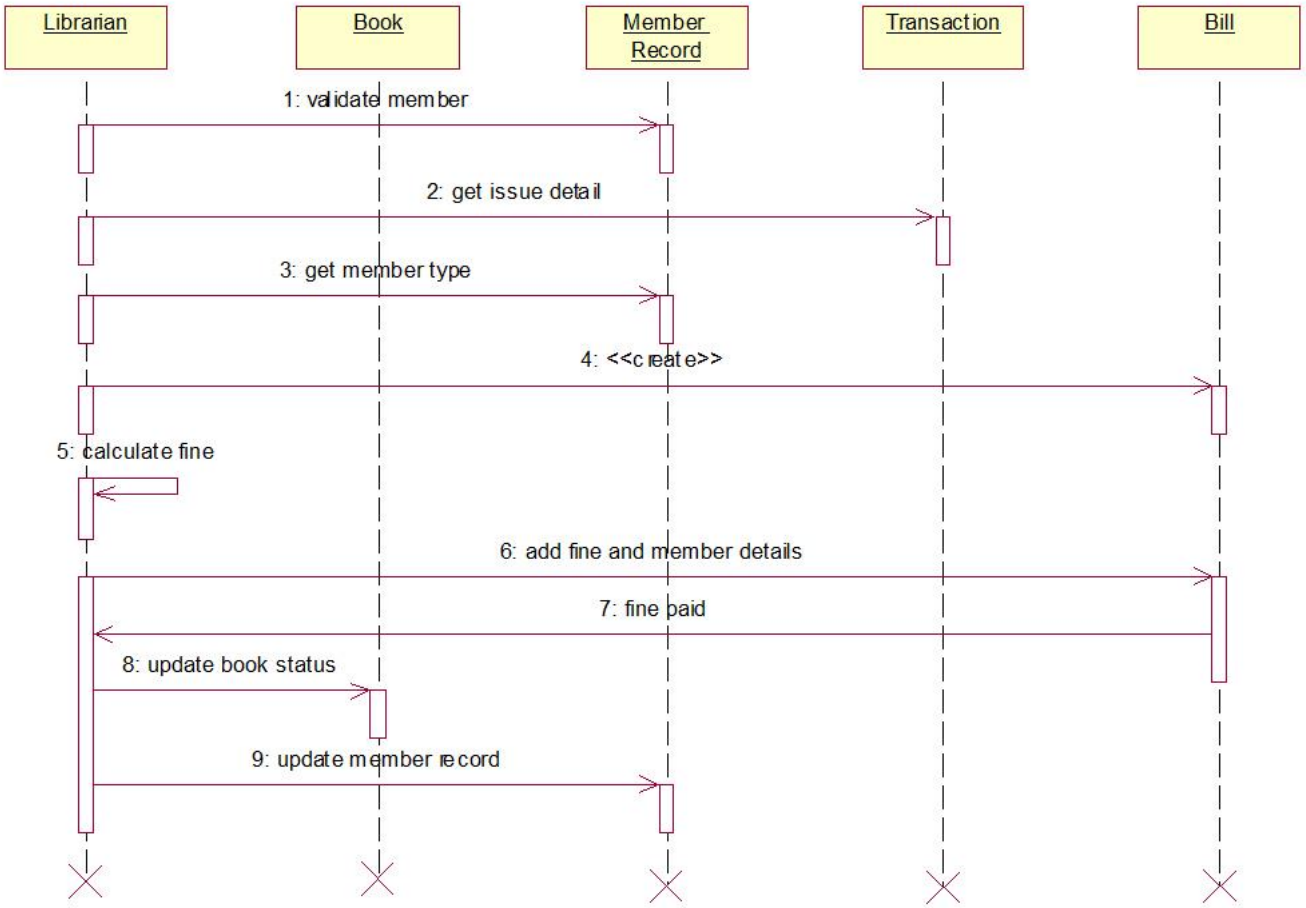
1.Use case diagram



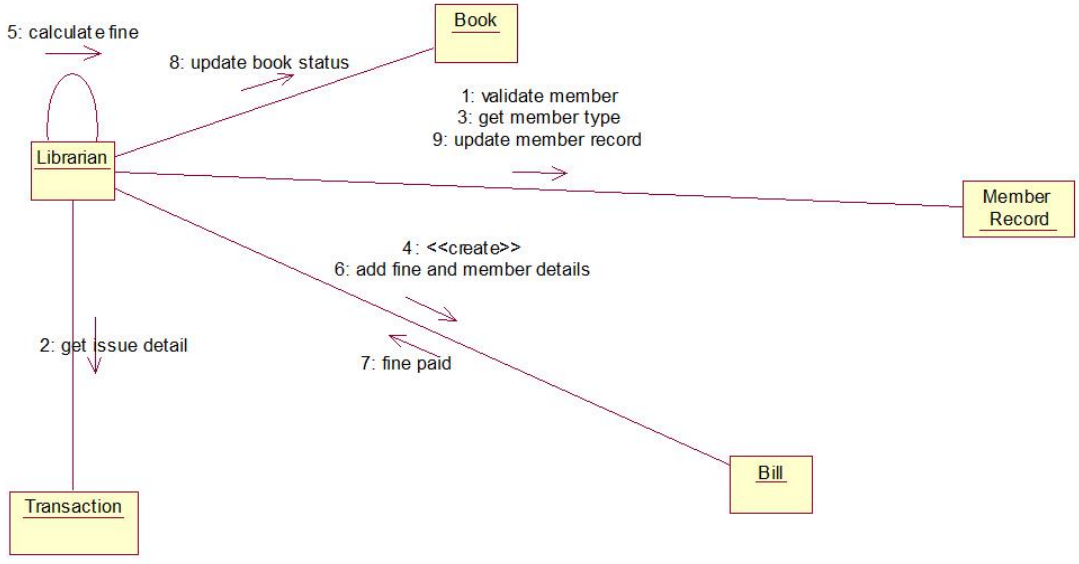
2.Class diagram



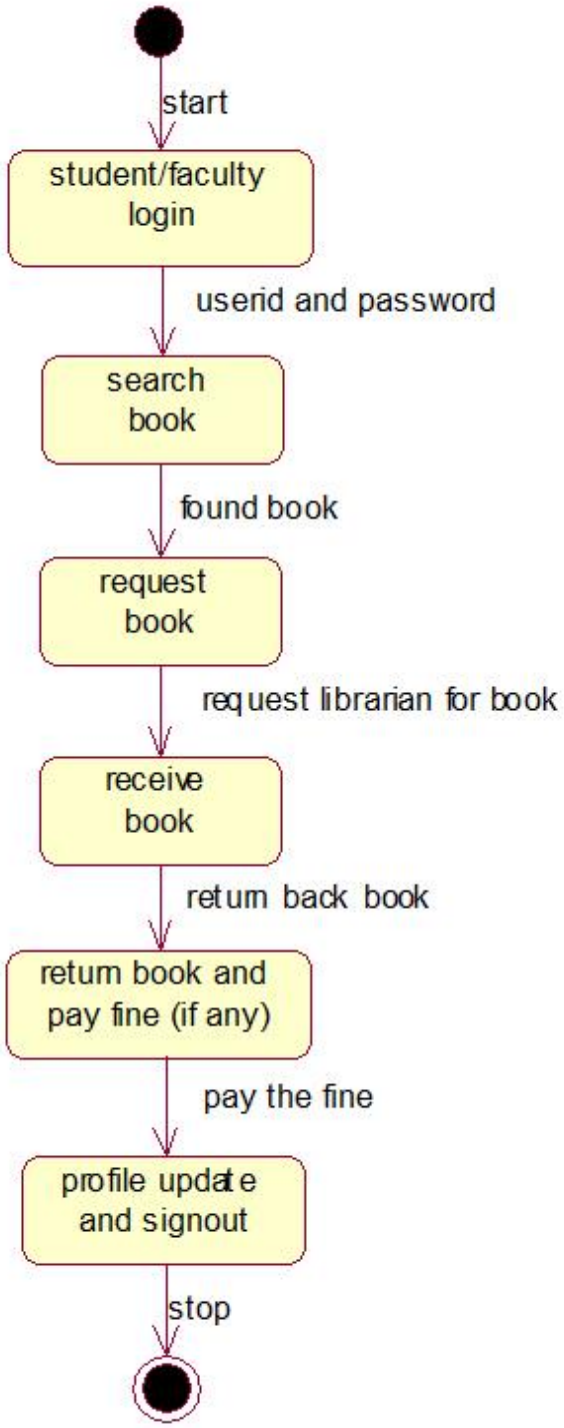
## 3.Sequence diagram



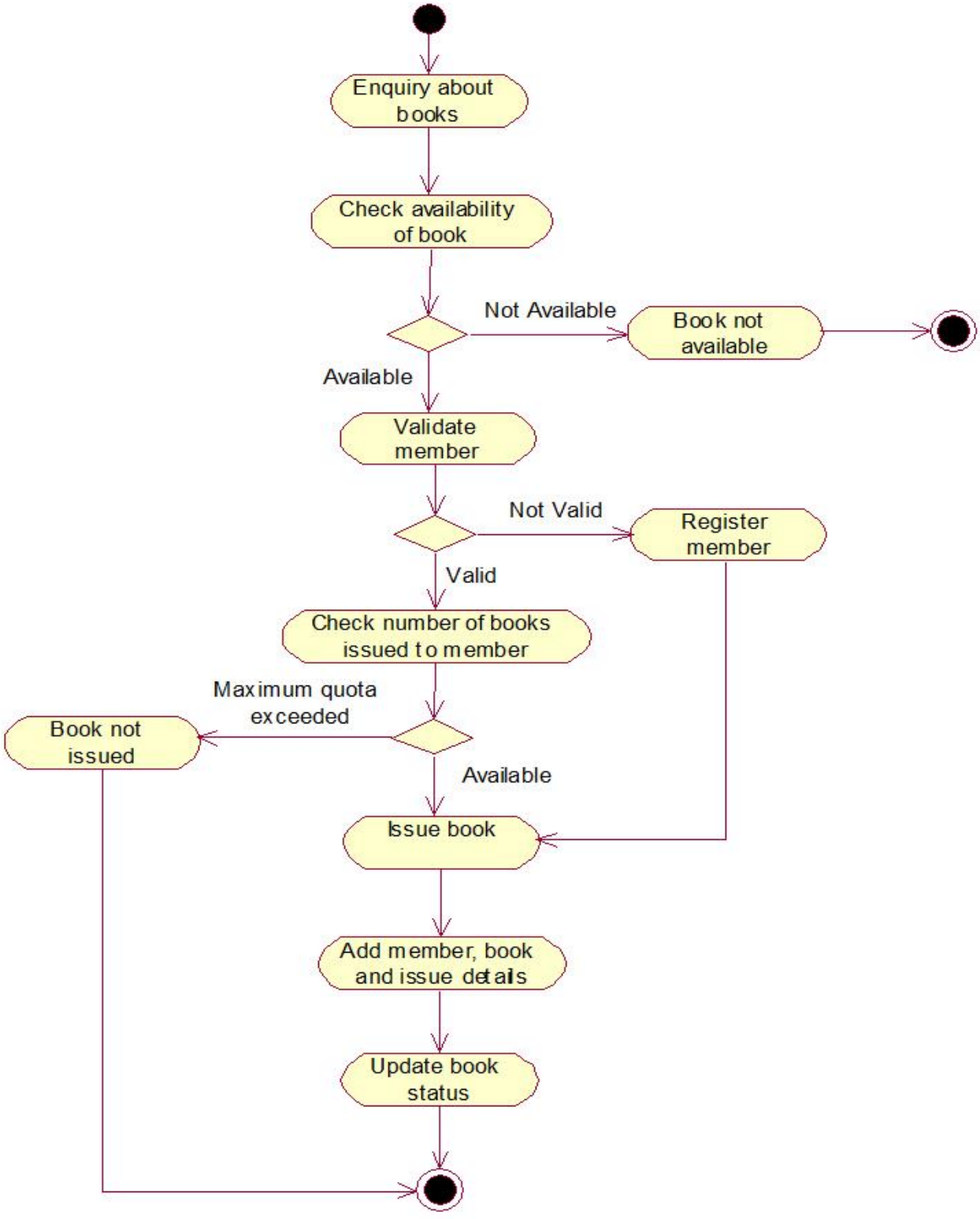

## 4.Collaboration diagram



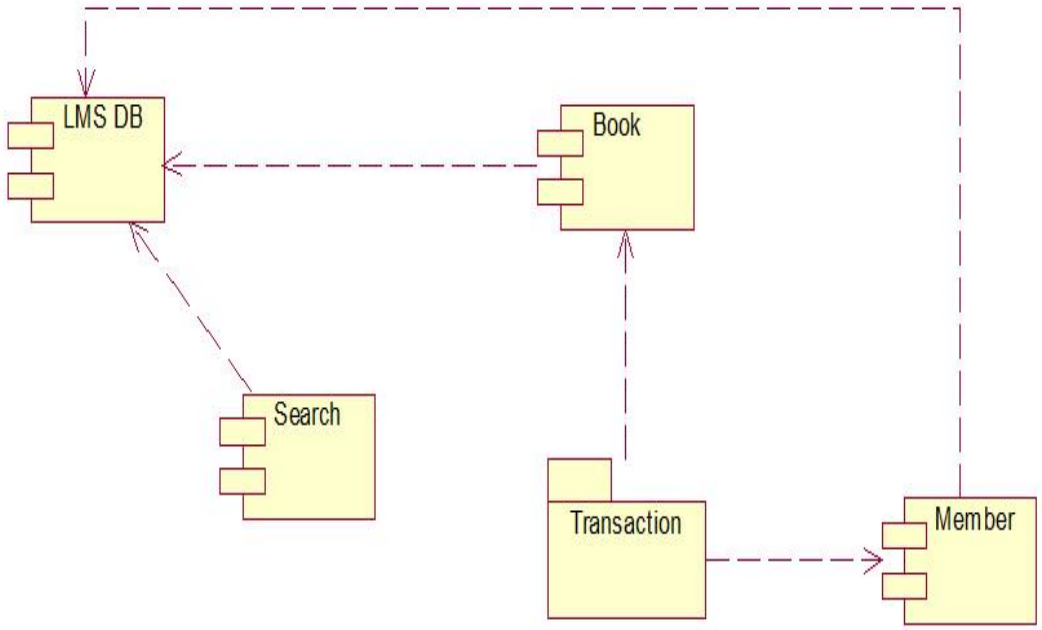
5.Statechart diagram



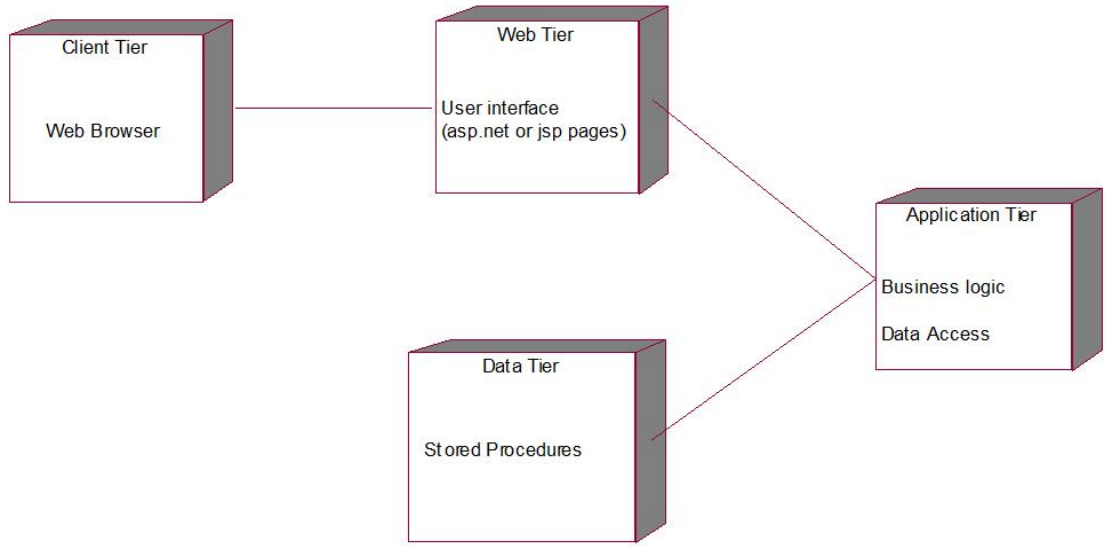
6.Activity diagram



7.Component diagram

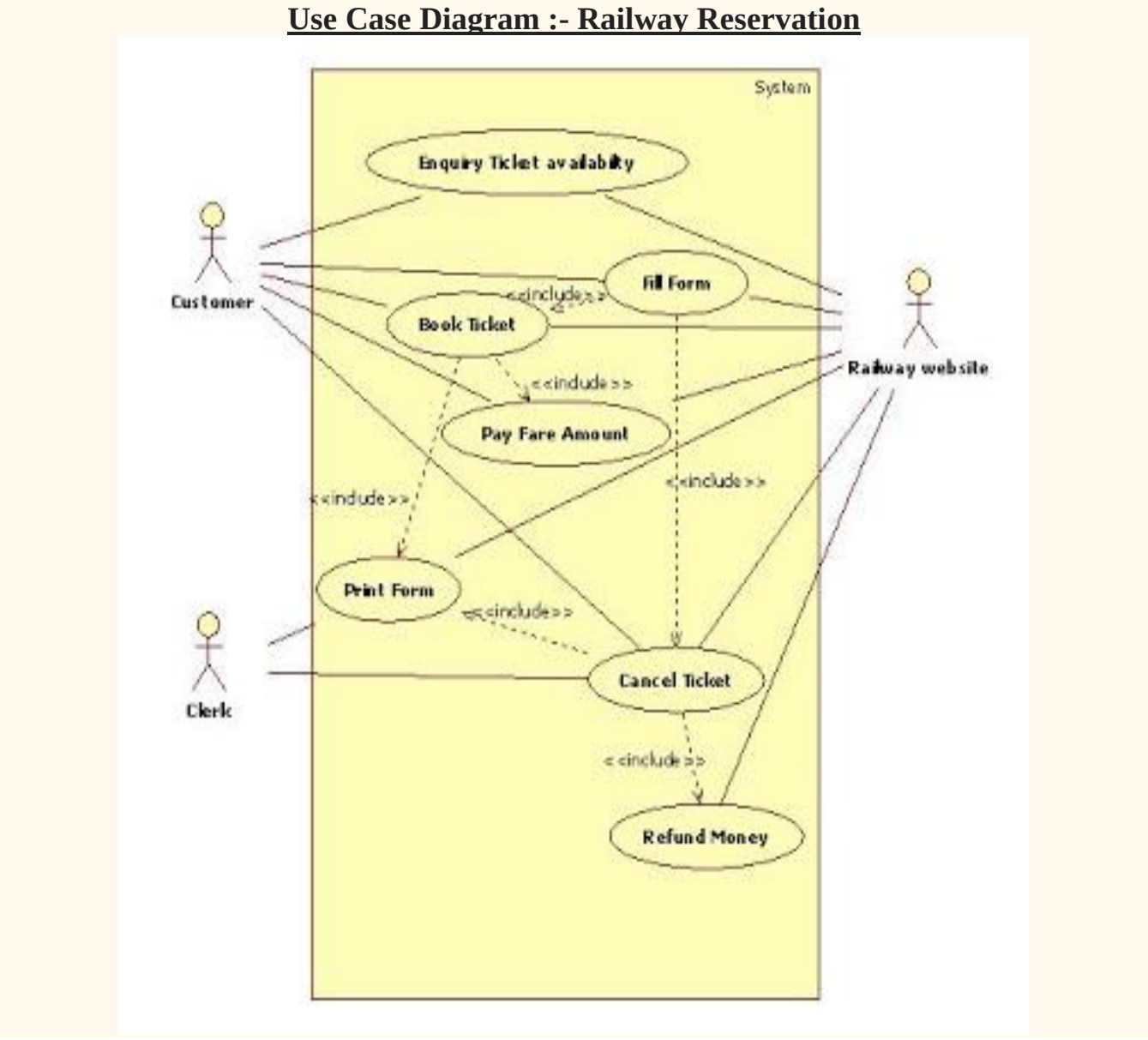



8.Deployment diagram

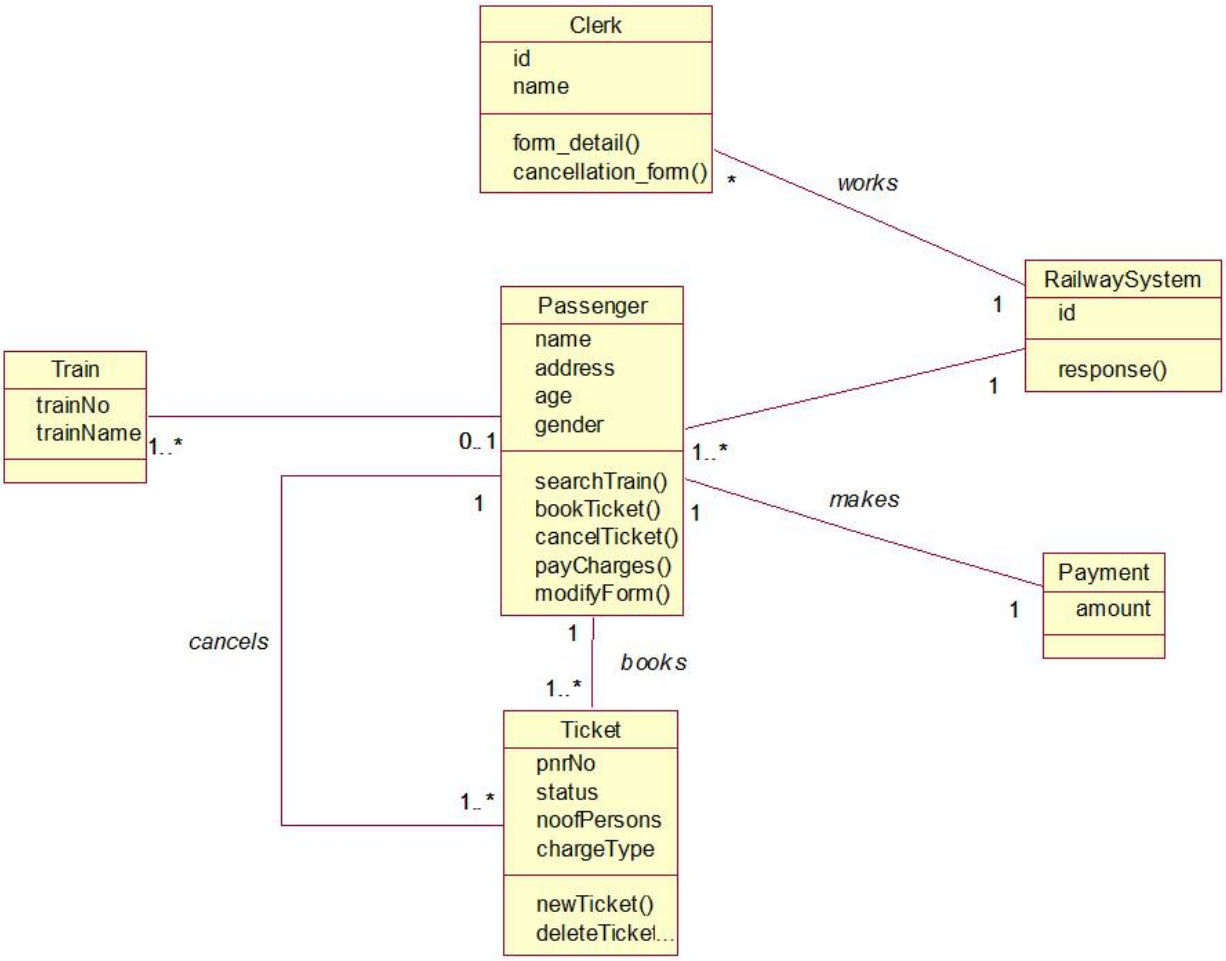


**Railway Reservation System**

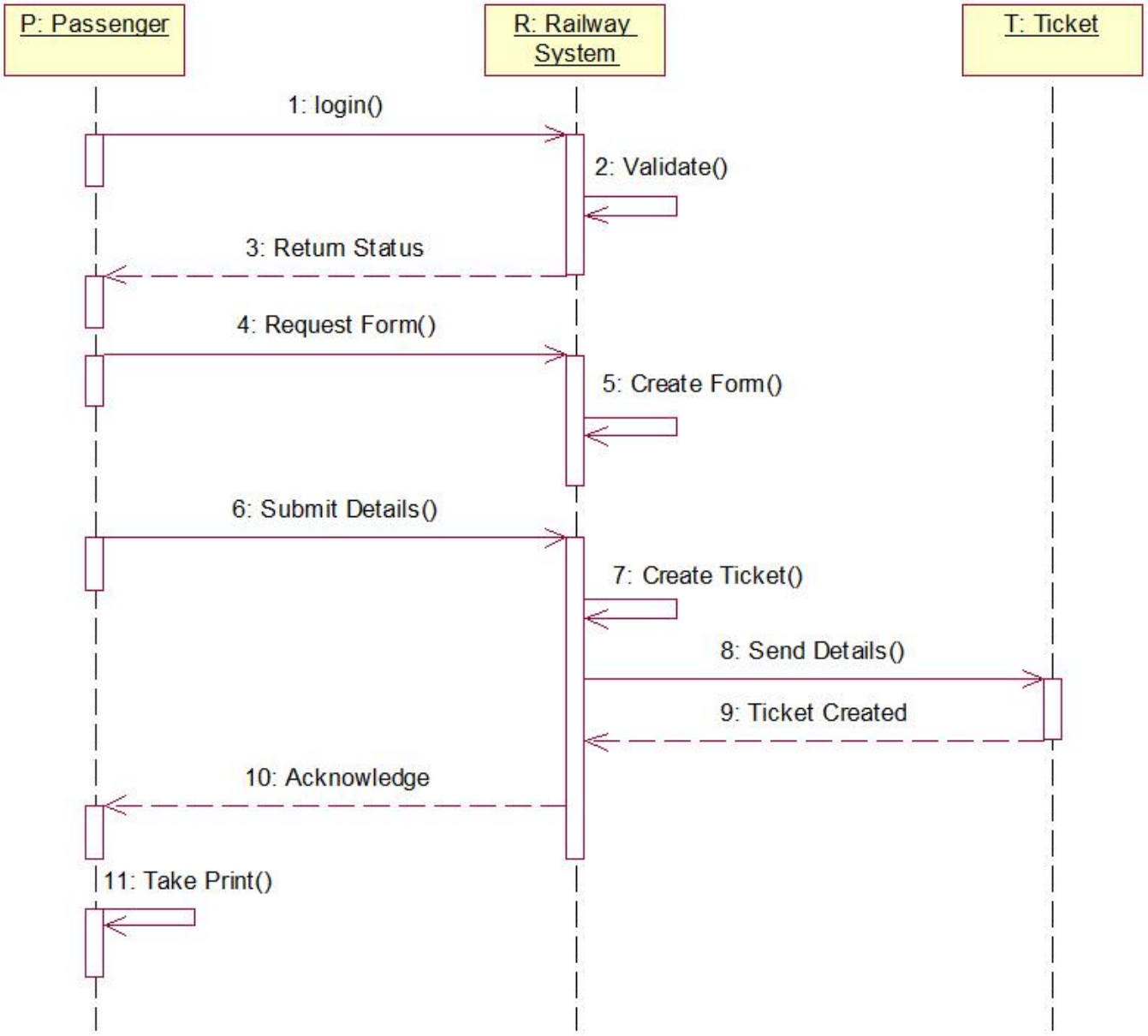
Use case diagram



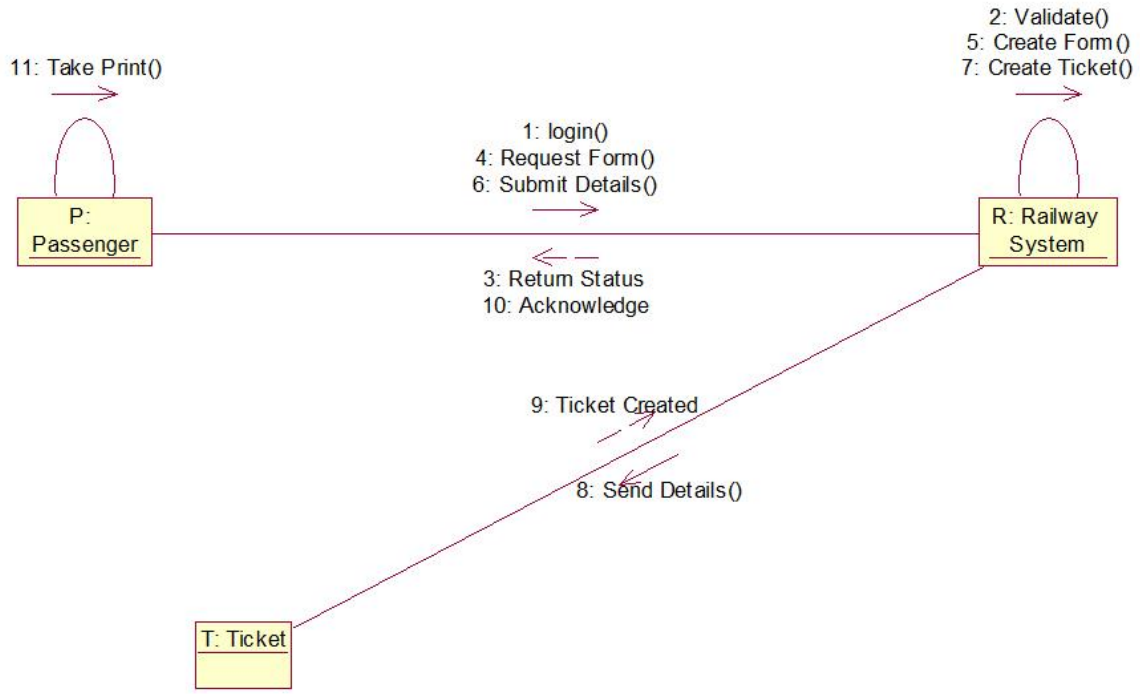
Class diagram



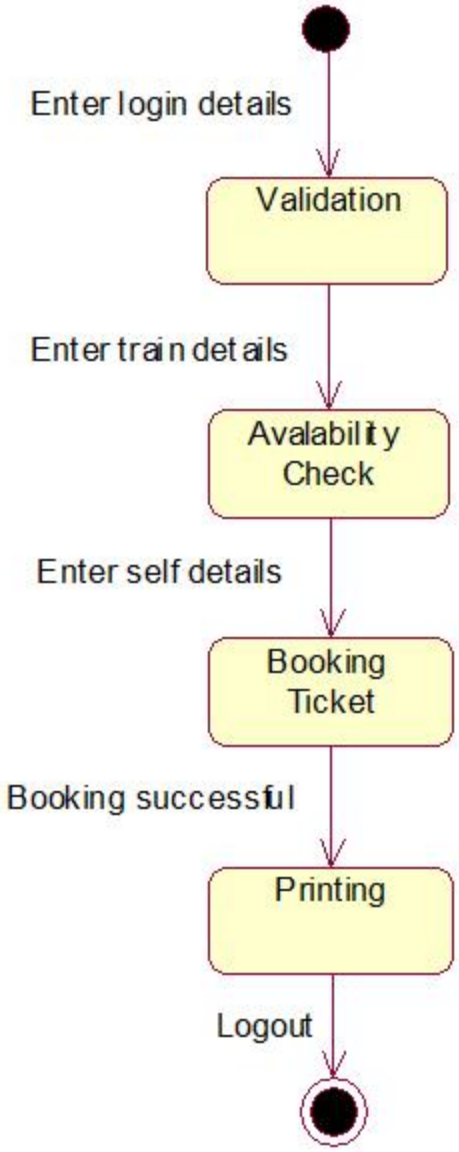
Sequence diagram



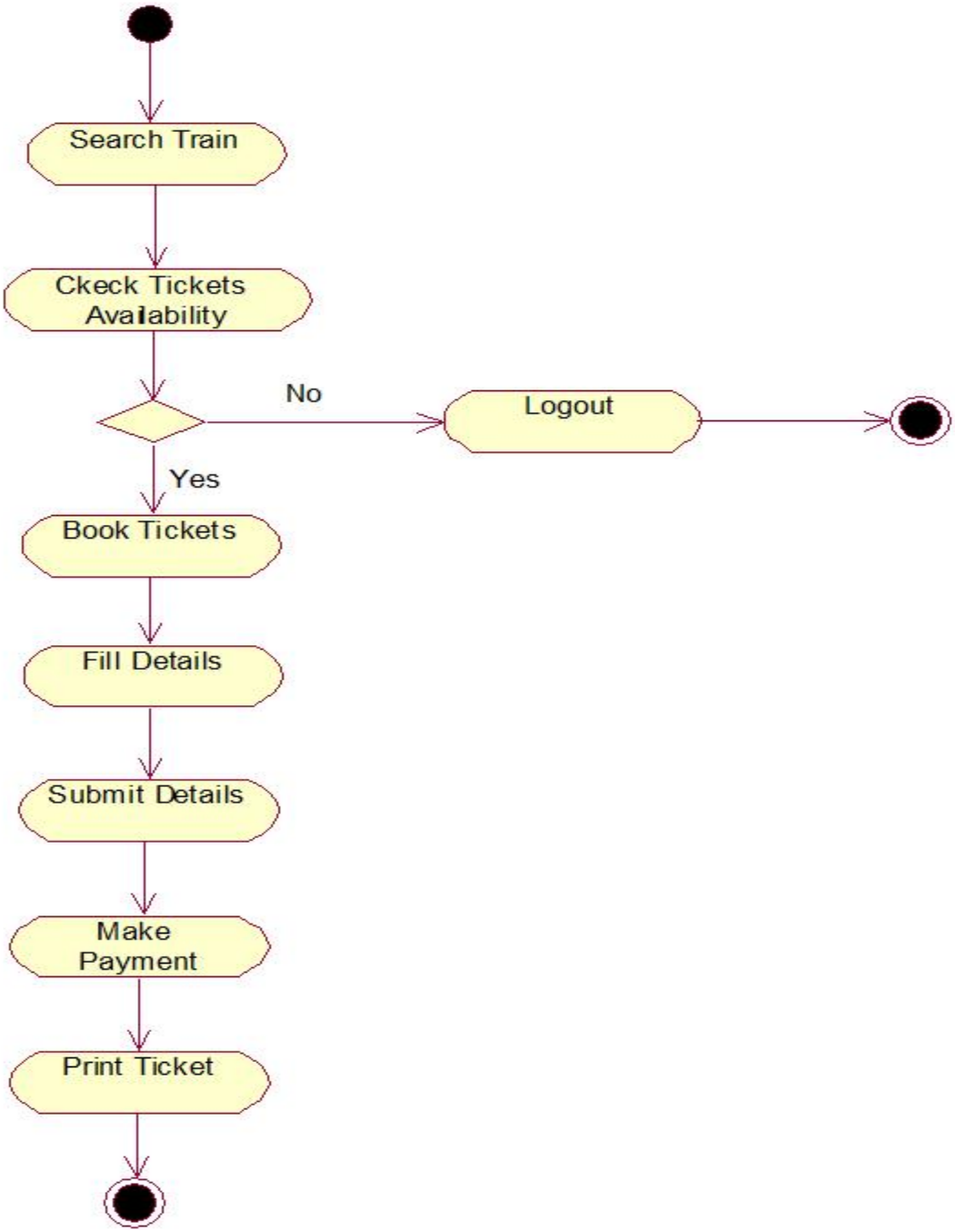
Collaboration diagram



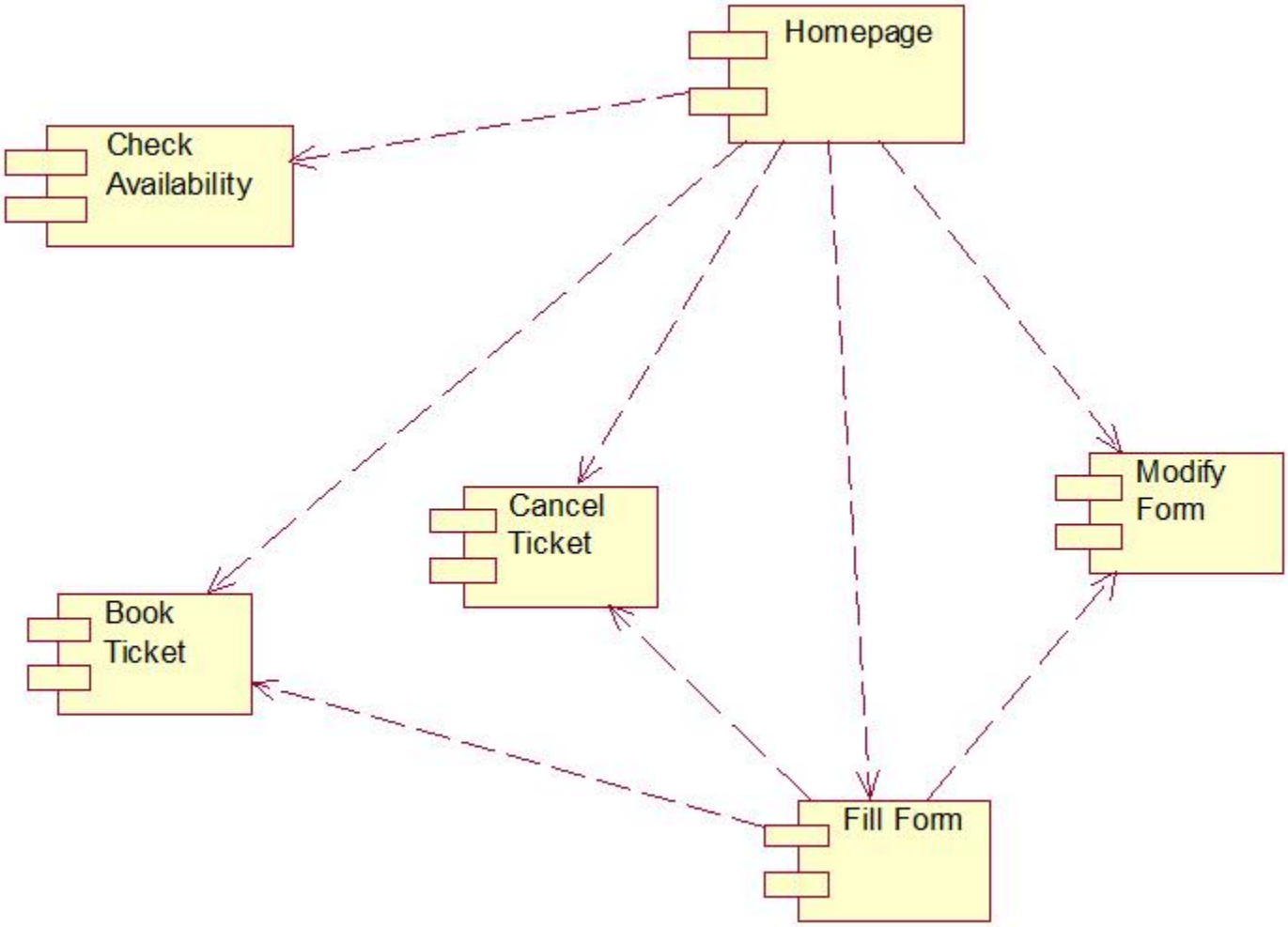
Statechart diagram



Activity diagram



**Component**


Homepage
Check Availability
Cancel Ticket
Modify Form
Book Ticket
Fill Form

Deployment diagram

