**SOFTWARE ENGINEERING LAB FILE**

****

**Amity school of engineering & Technology**

**Submitted to: - Submitted by: -**

**Dr. MANOJ KUMAR SHUKLA JAYANT BHALLA  
 A12405215021   
 5CSE – 1X (ASET)**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO. | LIST OF PROGRAMS | DATE | SIGNATURE |
| 1) | Introduction to UML |  |  |
| 2) | Use – Case Diagram |  |  |
| 3) | Class Diagram |  |  |
| 4) | State Chart Diagram |  |  |
| 5) | Activity Diagram |  |  |
| 6) | Sequence Diagram |  |  |

**EXPERIMENT – 1**

**OBJECTIVE:** Introduction to UML**.**

**EQUIPMENT / SOFTWARE USED:**

|  |  |  |
| --- | --- | --- |
| S.No. | Hardware | Software |
| 1. | I5 Processor | IBM Rational Rose |
| 2. | 8 GB RAM | Microsoft Word |
| 3. | Keyboard |  |
| 4. | Mouse |  |
| 5. | Monitor |  |
| 6. | Printer |  |

**THEORY:**

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

OMG is continuously making efforts to create a truly industry standard.

* UML stands for **Unified Modeling Language**.
* UML is different from the other common programming languages such as C++, Java, COBOL, etc.
* UML is a pictorial language used to make software blueprints.
* UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document software system.
* Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model non-software systems as well. For example, the process flow in a manufacturing unit, etc.

UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object oriented analysis and design. After some standardization, UML has become an OMG standard.

**Things**

**Things** are the most important building blocks of UML. Things can be −

* Structural
* Behavioral
* Grouping
* Annotational

**Structural Things**

**Structural things** define the static part of the model. They represent the physical and conceptual elements. Following are the brief descriptions of the structural things.

**Class −** Class represents a set of objects having similar responsibilities.

class

**Interface −** Interface defines a set of operations, which specify the responsibility of a class.

Interface

**Collaboration −**Collaboration defines an interaction between elements.

Collaboration

**Use case −**Use case represents a set of actions performed by a system for a specific goal.

Use case

**Component −**Component describes the physical part of a system.

Component

**Node −** A node can be defined as a physical element that exists at run time.



**Behavioral Things**

**A behavioral thing** consists of the dynamic parts of UML models. Following are the behavioral things −

**Interaction −** Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.

Interaction

**State machine −** State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change



**Grouping Things**

**Grouping things** can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available −

**Package −** Package is the only one grouping thing available for gathering structural and behavioral things.



**Annotational Things**

**Annotational things** can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. **Note** - It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.

Note

**Relationship**

**Relationship** is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.

There are four kinds of relationships available.

**Dependency**

Dependency is a relationship between two things in which change in one element also affects the other.

Dependency

**Association**

Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.

Association

**Generalization**

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects.

Generalization

**Realization**

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.

Realization

**UML Diagrams**

UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system.

The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete.

UML includes the following nine diagrams, the details of which are described in the subsequent chapters.

* Class diagram
* Object diagram
* Use case diagram
* Sequence diagram
* Collaboration diagram
* Activity diagram
* Statechart diagram
* Deployment diagram
* Component diagram

**EXPERIMENT – 2**

**OBJECTIVE:** To create a Use Case diagram for Railway Reservation System in UML.

**EQUIPMENT / SOFTWARE USED:**

|  |  |  |
| --- | --- | --- |
| S.No. | Hardware | Software |
| 1. | I5 Processor | IBM Rational Rose |
| 2. | 8GB RAM | Microsoft Word |
| 3. | Keyboard |  |
| 4. | Mouse |  |
| 5. | Monitor |  |
| 6. | Printer |  |

**THEORY:**

## What is a Use Case Diagram?

A use case diagram is a dynamic or behavior diagram in [UML](https://www.smartdraw.com/uml-diagram/). Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system.

## Why Make Use Case Diagrams?

Use case diagrams are valuable for visualizing the functional requirements of a system that will translate into design choices and development priorities.

They also help identify any internal or external factors that may influence the system and should be taken into consideration.

They provide a good high level analysis from outside the system. Use case diagrams specify how the system interacts with actors without worrying about the details of how that functionality is implemented.

## Basic Use Case Diagram Symbols and Notations

**System**   
Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.



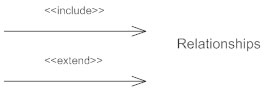
**Use Case**   
Draw use cases using ovals. Label the ovals with verbs that represent the system's functions.



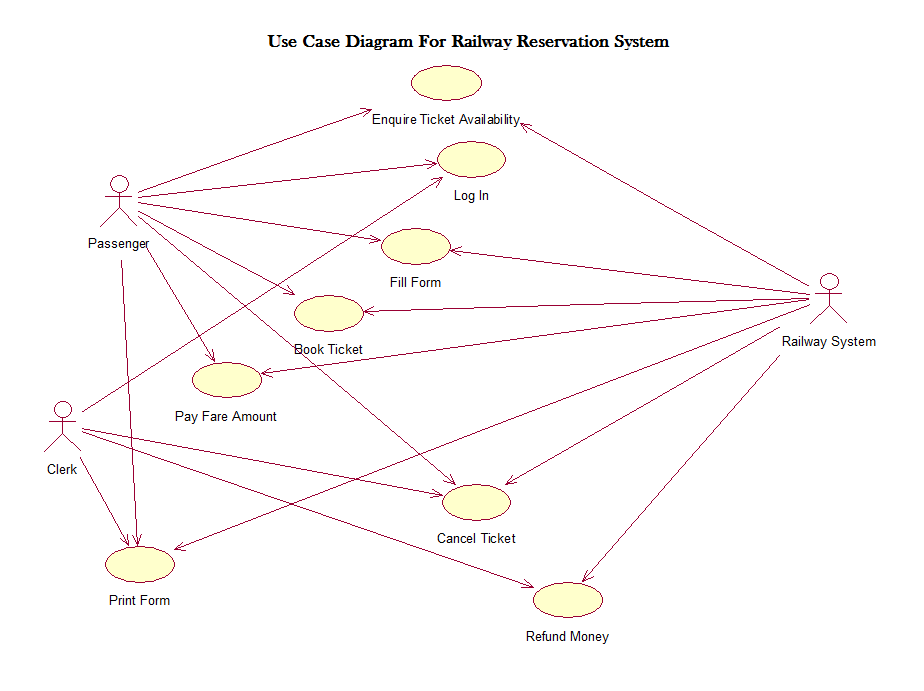
**Actors**   
Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.



**Relationships**   
Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.



**OUTPUT:**



**EXPERIMENT – 3**

**OBJECTIVE:** To create a Class diagram for ATM Management System in UML.

**EQUIPMENT / SOFTWARE USED:**

|  |  |  |
| --- | --- | --- |
| S.No. | Hardware | Software |
| 1. | I5 Processor | IBM Rational Rose |
| 2. | 8GB RAM | Microsoft Word |
| 3. | Keyboard |  |
| 4. | Mouse |  |
| 5. | Monitor |  |
| 6. | Printer |  |

**THEORY:**

## What is a Class Diagram?

A class diagram models the static structure of a system. It shows relationships between classes, objects, attributes, and operations.

## Basic Class Diagram Symbols and Notations

#### **Classes**

Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition (left-aligned, not bolded, and lowercase), and write operations into the third.



#### **Active Classes**

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.



#### **Visibility**

Use visibility markers to signify who can access the information contained within a class. Private visibility, denoted with a - sign, hides information from anything outside the class partition. Public visibility, denoted with a + sign, allows all other classes to view the marked information. Protected visibility, denoted with a # sign, allows child classes to access information they inherited from a parent class.



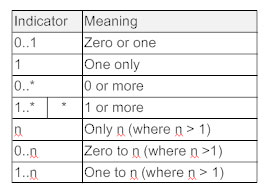
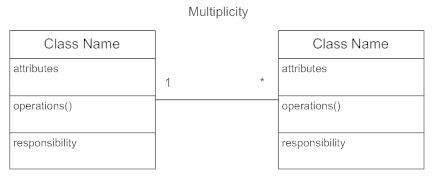
#### **Associations**

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.



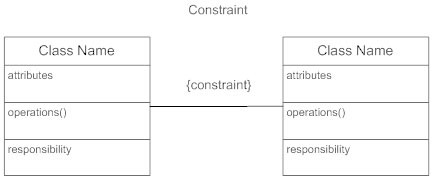
#### **Multiplicity (Cardinality)**

Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for just one company.



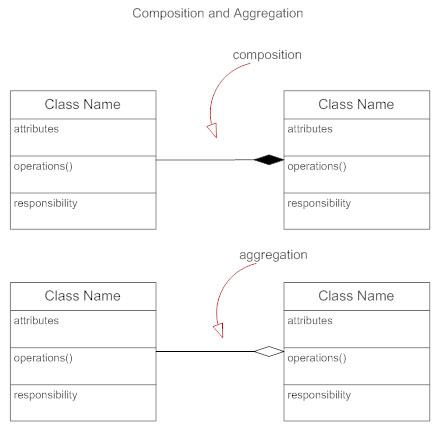
#### **Constraint**

Place constraints inside curly braces {}.



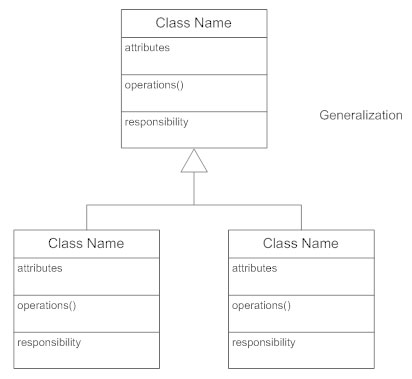
#### **Composition and Aggregation**

Composition is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part. Illustrate composition with a filled diamond. Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond ends in both composition and aggregation relationships point toward the "whole" class (i.e., the aggregation).



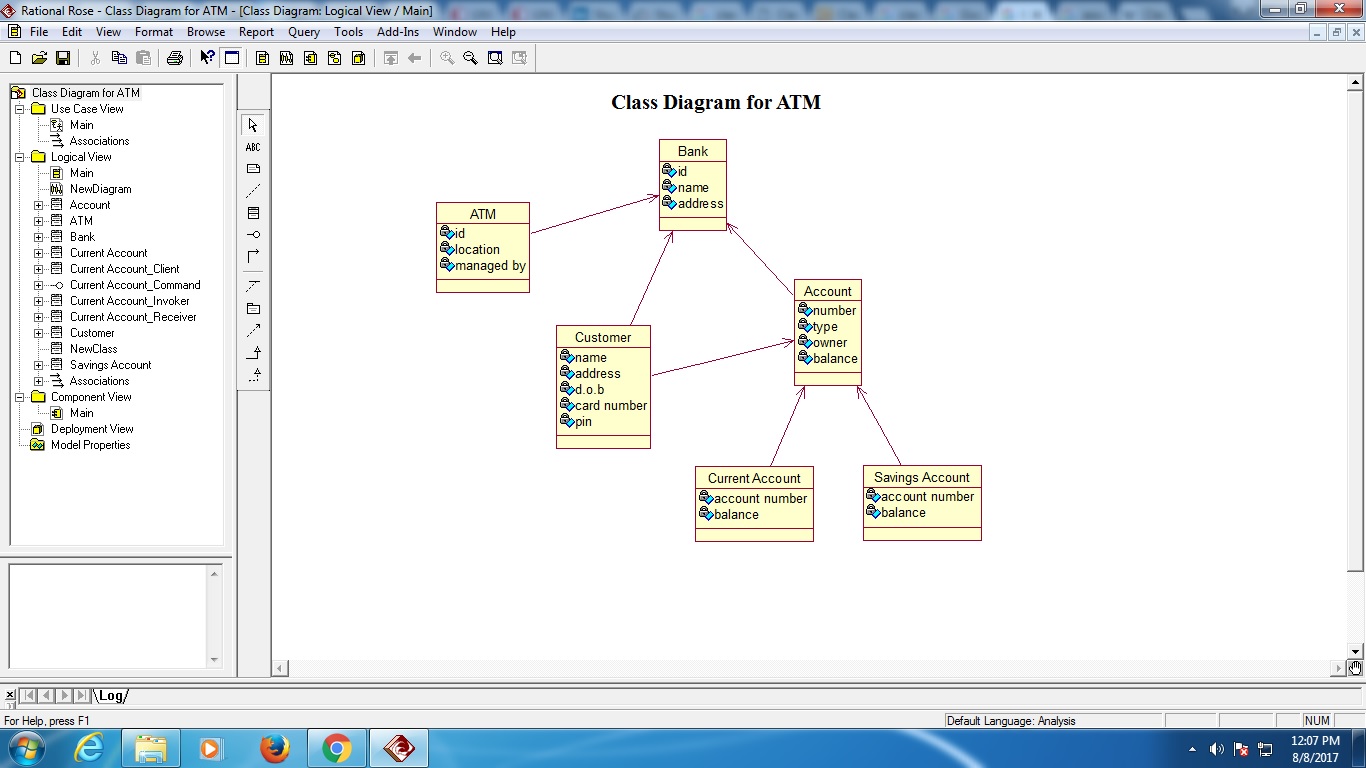
#### **Generalization**

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.



In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class. On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the superclass.

**OUTPUT;**



**EXPERIMENT – 4**

**OBJECTIVE:** To create a State diagram for Railway Reservation System in UML.

**EQUIPMENT / SOFTWARE USED:**

|  |  |  |
| --- | --- | --- |
| S.No. | Hardware | Software |
| 1. | I5 Processor | IBM Rational Rose |
| 2. | 8GB RAM | Microsoft Word |
| 3. | Keyboard |  |
| 4. | Mouse |  |
| 5. | Monitor |  |
| 6. | Printer |  |

**THEORY:**

## What is a State Diagram?

A state diagram shows the behavior of classes in response to external stimuli. Specifically a state diagram describes the behavior of a single object in response to a series of events in a system. Sometimes it's also known as a Harel state chart or a state machine diagram. This UML diagram models the dynamic flow of control from state to state of a particular object within a system.

## What is the Difference between a State Diagram and a Flowchart?

A flowchart illustrates processes that are executed in the system that change the state of objects. A state diagram shows the actual changes in state, not the processes or commands that created those changes.

## How to Draw a State Diagram

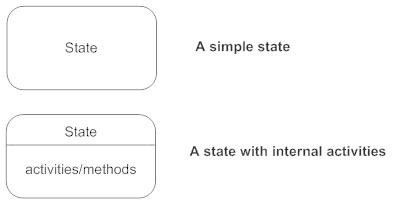
Before you begin your drawing find the initial and final state of the object in question.

Next, think of the states the object might undergo. For example, in e-commerce a product will have a release or available date, a sold out state, a restocked state, placed in cart state, a saved on wish list state, a purchased state, and so on.

Certain transitions will not be applicable when an object is in a particular state, for example a product can be in a purchased state or a saved in cart state if its previous state is sold out.

## Basic State Chart Diagram Symbols and Notations

**States**  
**States** represent situations during the life of an object. You can easily illustrate a state in SmartDraw by 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.

Transitions - State diagram

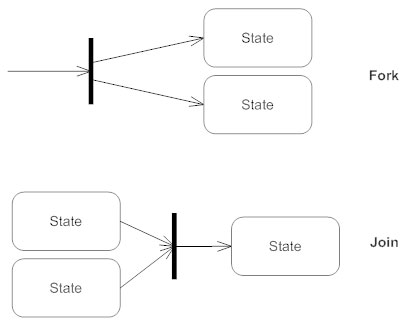
**Initial State**  
A filled circle followed by an arrow represents the object's initial state.

Inititial state - State diagram

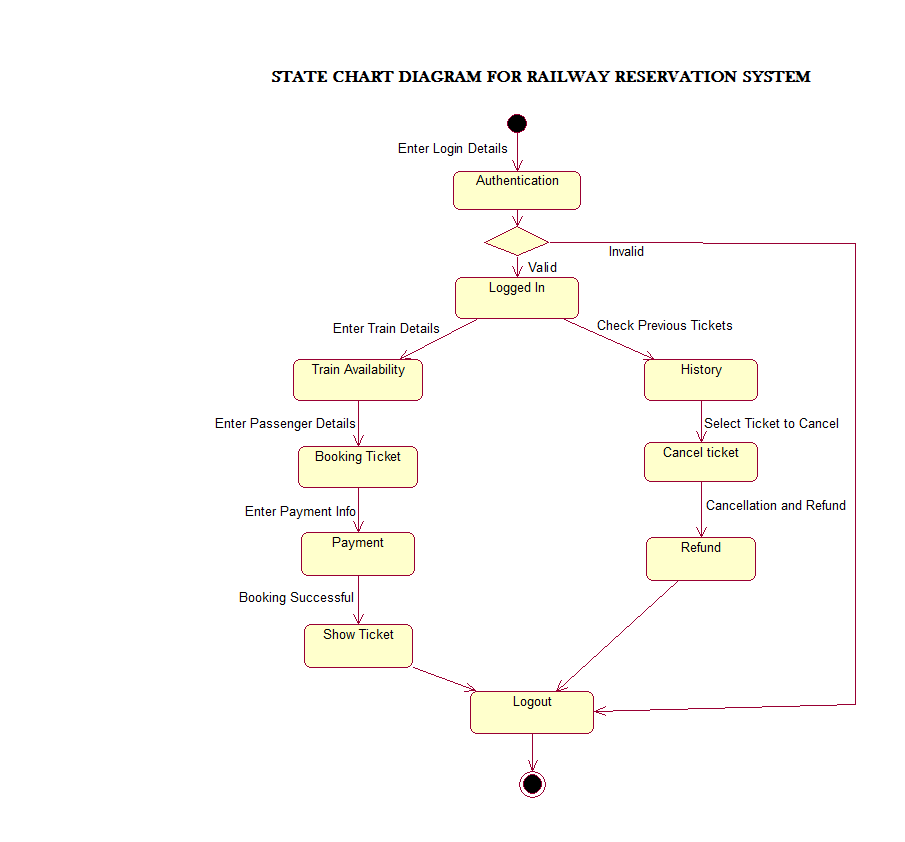
**Final State**  
An arrow pointing to a filled circle nested inside another circle represents the object's final state.

Final state - State diagram

**Synchronization and Splitting of Control**  
A short heavy bar with two transitions entering it represents a synchronization of control. The first bar is often called a fork where a single transition splits into concurrent multiple transitions. The second bar is called a join, where the concurrent transitions reduce back to one.



**OUTPUT:**



**EXPERIMENT – 5**

**OBJECTIVE:** To Create an Activity diagram for Railway Reservation System in UML.

**EQUIPMENT / SOFTWARE USED:**

|  |  |  |
| --- | --- | --- |
| S.No. | Hardware | Software |
| 1. | I5 Processor | IBM Rational Rose |
| 2. | 8GB RAM | Microsoft Word |
| 3. | Keyboard |  |
| 4. | Mouse |  |
| 5. | Monitor |  |
| 6. | Printer |  |

**THEORY:**

## What is an Activity Diagram?

An activity diagram visually presents a series of actions or flow of control in a system similar to a [flowchart](https://www.smartdraw.com/flowchart/) or a [data flow diagram](https://www.smartdraw.com/data-flow-diagram/). Activity diagrams are often used in business process modeling. They can also describe the steps in a [use case diagram](https://www.smartdraw.com/use-case-diagram/). Activities modeled can be sequential and concurrent. In both cases an activity diagram will have a beginning and an

## Basic Activity Diagram Notations and Symbols

##### **Initial State or Start Point**

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 symbol - Activity diagram

##### **Activity 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.



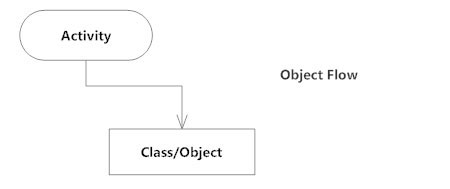
##### **Action Flow**

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 - Activity diagram

##### **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.



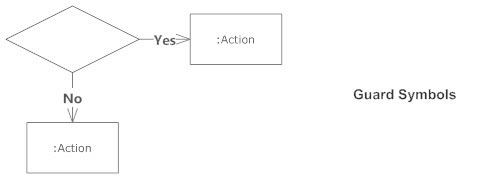
##### **Decisions and Branching**

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."



##### **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.

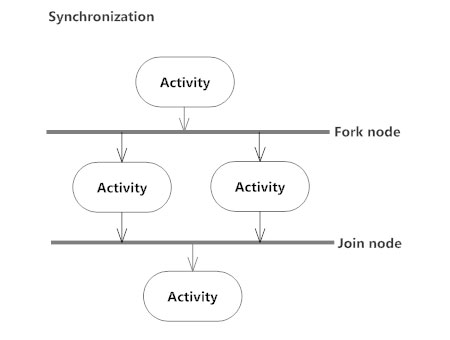


##### **Synchronization**

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.

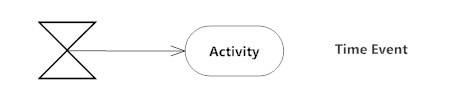
A join node joins multiple concurrent flows back into a single outgoing flow.

A fork and join mode used together are often referred to as synchronization.



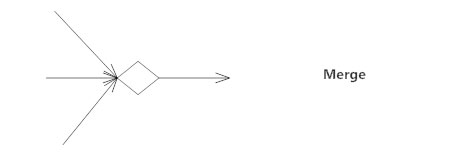
##### **Time Event**

This refers to an event that stops the flow for a time; an hourglass depicts it.



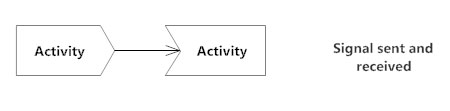
##### **Merge Event**

A merge event brings together multiple flows that are not concurrent.



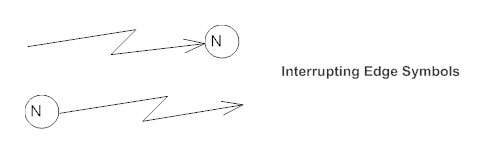
##### **Sent and Received Signals**

Signals represent how activities can be modified from outside the system. They usually appear in pairs of sent and received signals, because the state can't change until a response is received, much like synchronous messages in a [sequence diagram](https://www.smartdraw.com/sequence-diagram/). For example, an authorization of payment is needed before an order can be completed.



##### **Interrupting Edge**

An event, such as a cancellation, that interrupts the flow denoted with a lightning bolt.



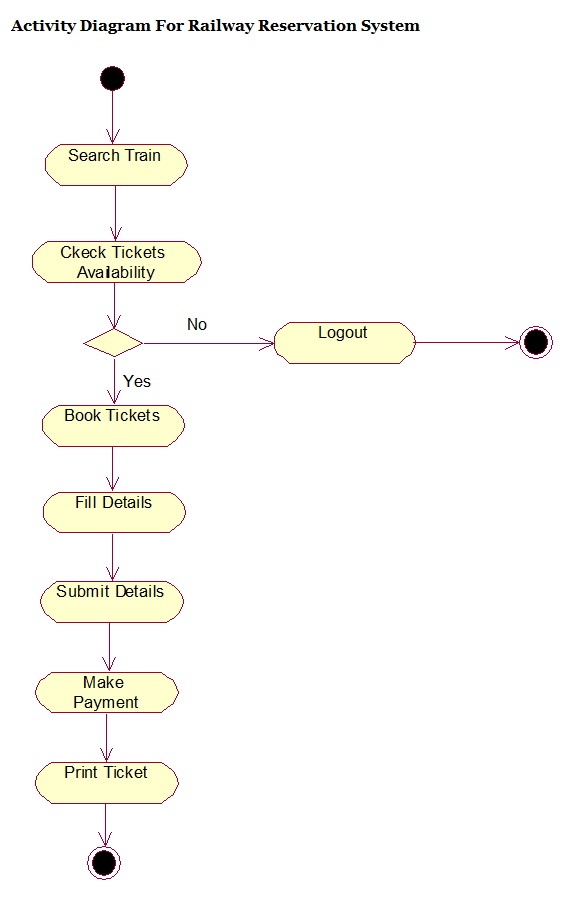
##### **Swimlanes**

Swimlanes group related activities into one column.

##### **Final State or End Point**

An arrow pointing to a filled circle nested inside another circle represents the final action state.

**OUTPUT:**



**EXPERIMENT – 6**

**OBJECTIVE:** To create a Sequence diagram for Railway Reservation System in UML.

**EQUIPMENT / SOFTWARE USED:**

|  |  |  |
| --- | --- | --- |
| S.No. | Hardware | Software |
| 1. | I5 Processor | IBM Rational Rose |
| 2. | 8GB RAM | Microsoft Word |
| 3. | Keyboard |  |
| 4. | Mouse |  |
| 5. | Monitor |  |
| 6. | Printer |  |

**THEORY:**

## What is a Sequence Diagram?

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. These can help to predict how a system will behave and to discover responsibilities a class may need to have in the process of modeling a new system.

## Basic Sequence Diagram Notations

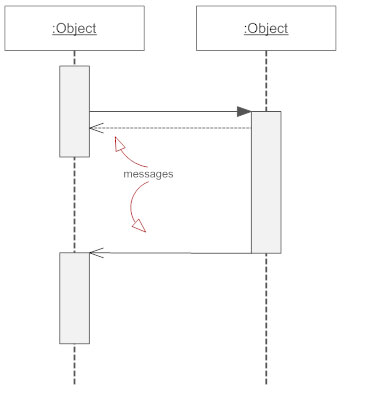
**Class Roles or Participants**  
Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

Object symbol - Sequence diagram

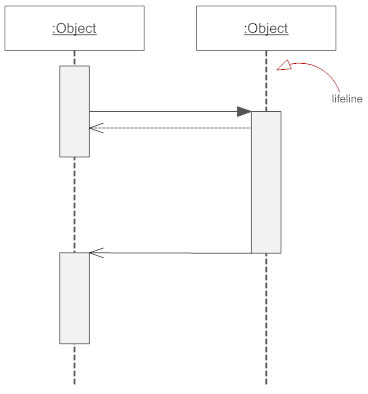
**Activation or Execution Occurrence**  
Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.



**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.

Synchronous message - Sequence diagram

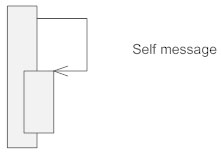
**Asynchronous Message**  
Asynchronous messages don't need a reply for interaction to continue. Like synchronous messages, they are drawn with an arrow connecting two lifelines; however, the arrowhead is usually open and there's no return message depicted.

Simple messageAsyncrhonous message

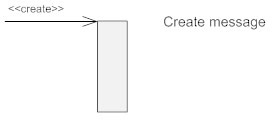
**Reply or Return Message**  
A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

Reply messages

**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.

Delete message

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

Found message

**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.

Lost message

**OUTPUT:**

