MODULE- 1

Chapter-1

Approaches to Software Design

Software

Software is a collection of instructions that enable the user to interact with a computer , its hardware or perform tasks.Without software, most computers would be useless. For example, without your Internet browser software, you could not surf the Internet. Without an operating system, the browser could not run on your computer.

There are two types of software

1. System Software

2. Application Software

Examples of system software are Operating System, Compilers, Interpreter, Assemblers, etc.

Examples of Application software are Railways Reservation Software, Microsoft Office Suite Software, Microsoft Word, Microsoft PowerPoint , etc.

Software Design

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. The design process for software systems often has two levels. At the first level the focus is on deciding which modules are needed for the system on the basis of SRS (Software Requirement Specification) and how the modules should be interconnected. Software design is the first step in SDLC (Software Design Life Cycle) It tries to specify how to fulfill the requirements mentioned in SRS document.

Functional Oriented Design (FOD)

In function-oriented design, the system consists of many smaller sub-systems known as **functions.** These functions are capable of performing significant task in the system Function oriented design inherits some properties of structured design where divide and conquer methodology is used. This design mechanism divides the whole system into smaller functions.These functional modules can share information among themselves by means of information passing and using information available globally.

Eg: Banking process

Here withdraw, Deposit, Transfer are functions and that can be divided in to sub- functions again. So, in FOD, the entire problem is divided in to number of functions and those functions are broken down in to smaller functions and these smaller functions are converted in to software modules.

Object Oriented Design (OOD)

OOD is based on Objects and interaction between the objects. Interaction between objects is called message **communication**. It involves the designing of **Objects, Classes** and the relationship between the classes

Consider the previous example of Banking process. Here, customer, money and account are objects

In OOD, implementation of a software based on the **concepts of objects.**

This approach is very close to the real-world applications

**Basic Object Oriented concepts

OBJECT:

Objects are real-world entities that has their own properties and behavior. It has physical existence

Eg: person, banks, company, customers etc

CLASS:

A class is a blueprint or prototype from which objects are created. A class is a generalized description of an object. An object is an instance of a class.

Relationship between Object & Class:

Let's take Human Being as a class. My name is John, and I am an instance/object of the class Human Being

Object has a physical existence while a class is just a logical definition.

ENCAPSULATION:

The wrapping up of data(variables) and function (methods) into a single unit (called class) is known as **encapsulation.** It is also called "**information hiding**"

Key Points of Encapsulation:-

• Protection of data from accidental corruption

• Flexibility and extensibility of the code and reduction in complexity

• Encapsulation of a class can hide the internal details of how an object does something

• Encapsulation protects abstraction

ABSTRACTION:

Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car.

POLYMORPHISM

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Eg: A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called **polymorphism.**

An operation may exhibit different behaviors in different instances. The behavior depends upon the types of data used in the operation.

INHERITANCE

The capability of a class to derive properties and characteristics from another class is called **Inheritance.**

Inheritance is the process by which objects of one class acquired the properties of objects of another classes

Sub Class : The class that inherits properties from another class is called **Subclass or Derived Class.**

Super Class : The class whose properties are inherited by sub class is called Base Class or Super class.

REUSABILITY: Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already aclass that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Unified Modeling Language (UML)

UML (Unified Modeling Language) is a general-purpose, graphical modeling language in the field of Software Engineering. UML is used to specify, visualize, construct, and document the artifacts (major elements) of the software system. UML is a visual language for developing software blueprints (designs). A blueprint or design represents the model.

For example, while constructing buildings, a designer or architect develops the building blueprints. Similarly, we can also develop blueprints for a software system.

UML is the most commonly and frequently used language for building software system blueprints. UML is not a programming language, it is rather a visual language.

The UML has the following features:

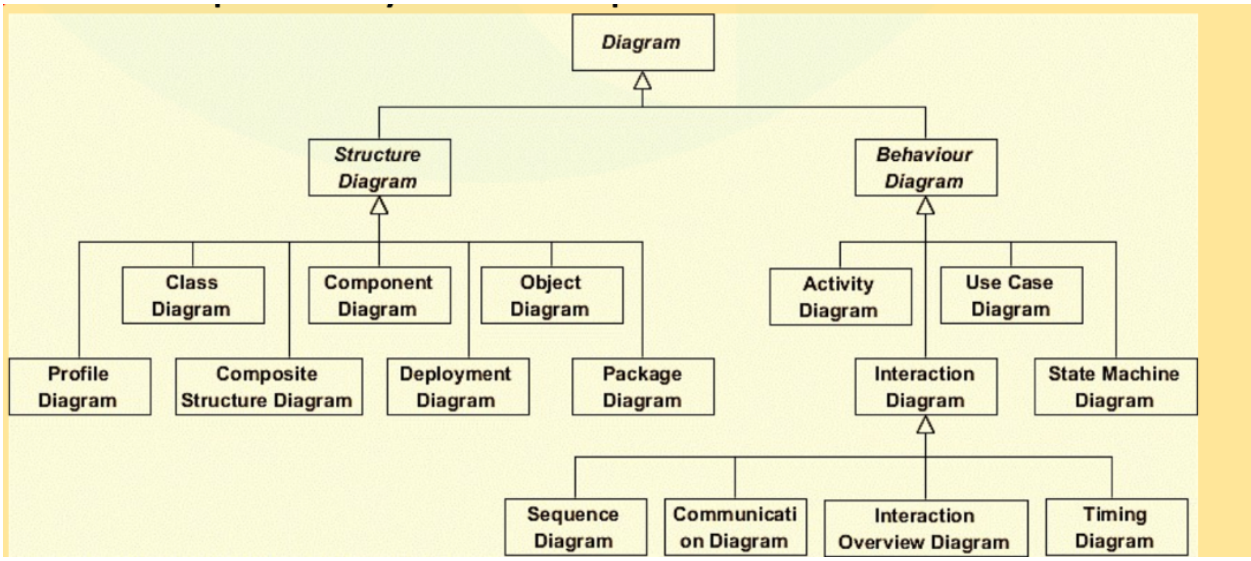
- It is a generalized modeling language.
- It is distinct from other programming languages like C++, Python, etc.

- It is interrelated to object-oriented analysis and design.
- It is used to visualize the workflow of the system.
- It is a pictorial language, used to generate powerful modeling artifacts

UML is linked with object oriented design and analysis

Diagrams in UML can be broadly classified as:

1. **Structure Diagrams** : Capture static aspects or structure of a system
2. **Behavior Diagrams**: Capture dynamic aspects or behavior of the system



1. CLASS DIAGRAM

The most widely used UML diagram is the **class diagram.** It is the building block of all **object oriented software systems**. Using class diagrams we can create the static structure of a system by showing the system's classes, their methods and attributes. Class diagrams also help us identify **relationships between different classes or objects.**

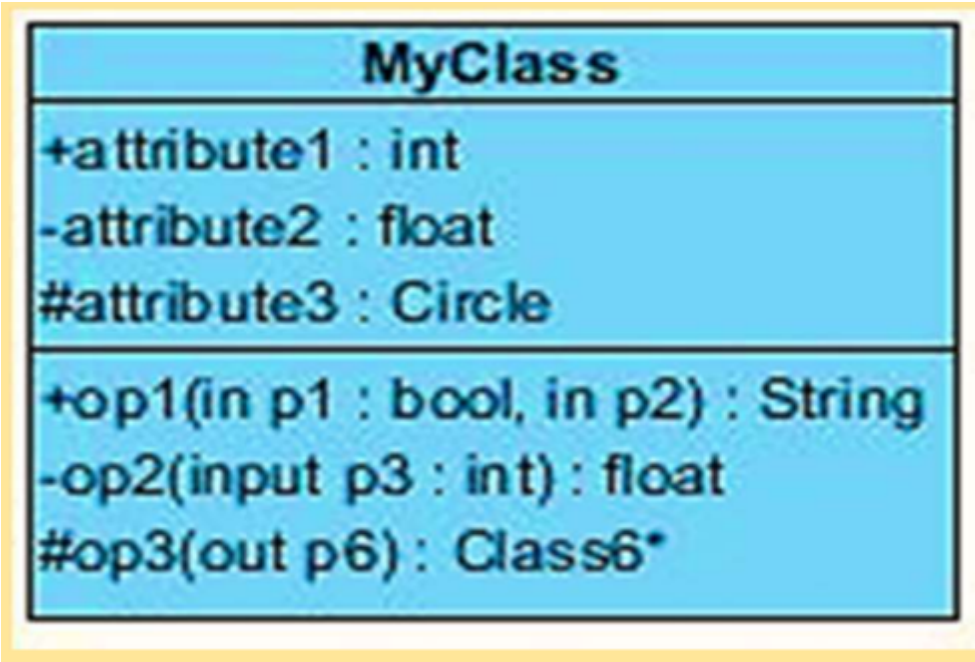
There are several software available which can be used online and offline to draw these diagrams Like **Edraw max, lucid chart** etc.

Class Notation:

A class notation consists of three parts:

1. **Class Name:**
   The name of the class appears in the first partition.
2. **Class Attributes:**
   Attributes are shown in the second partition.
   The attribute type is shown after the colon.
   Attributes map onto member variables (data members) in code.
3. **Class Operations (Methods):**
   Operations are shown in the third partition.
   They are services the class provides.

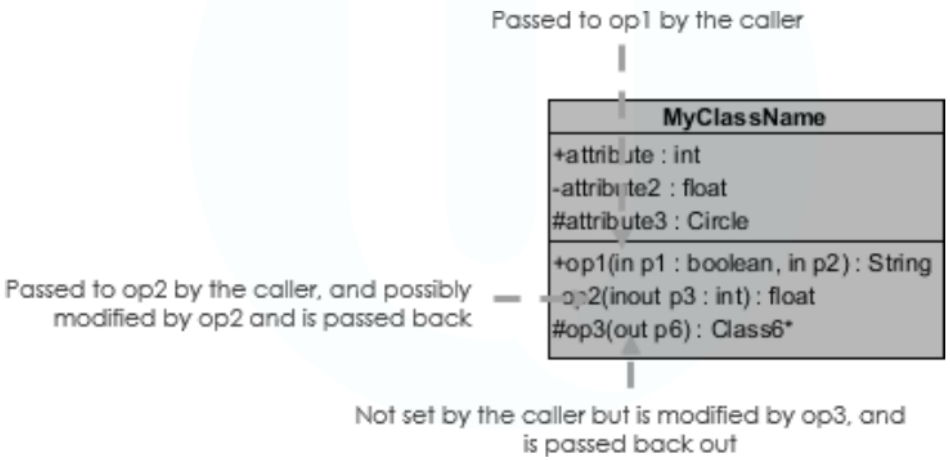
The return type of a method is shown after the colon at the end of the method signature.
The return type of method parameters are shown after the colon following the parameter name.
Operations map onto class methods in code



Class Visibility:
The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations

Parameter Directionality:
Each parameter in an operation (method) may be denoted as in, out or in out which specifies its direction with respect to the caller. This directionality is shown before the parameter name.

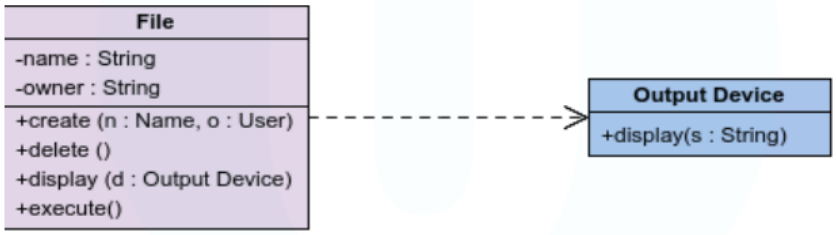


Relationships Between Classes:

UML is not just about pretty pictures. If used correctly, UML precisely conveys how code should be implemented from diagrams. If precisely interpreted, the implemented code will correctly reflect the intent of the designer.

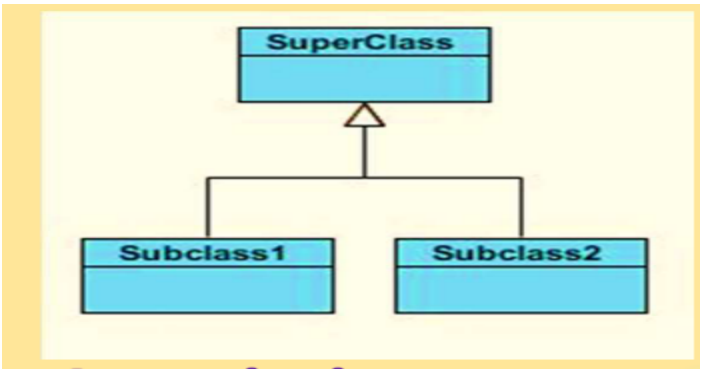
1) **Dependency**

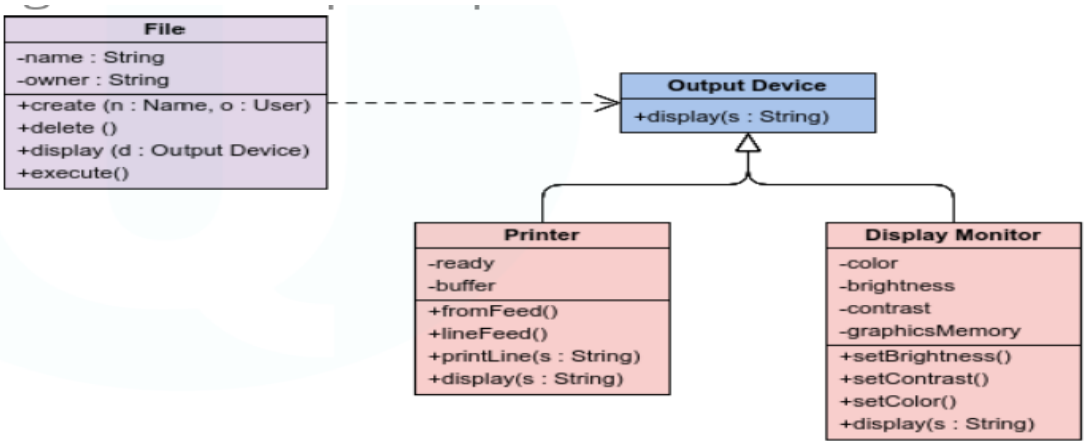
- A dependency means the relation between two or more classes in which a change in one may force changes in the other.
- Dependency indicates that one class depends on another.
- A dashed line with an open arrow.



2)Inheritance (or Generalization)

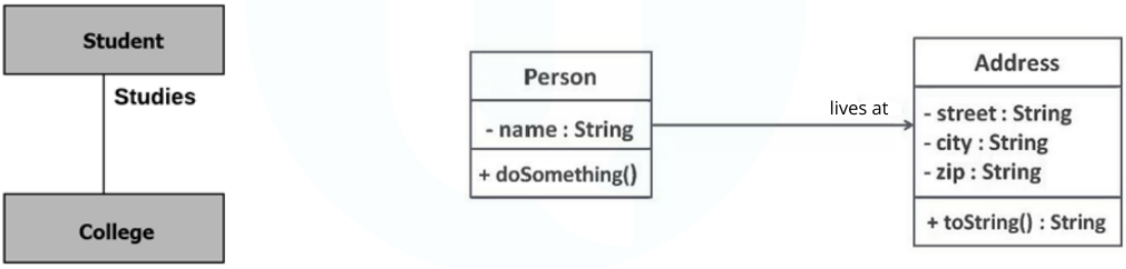
- A **generalization** helps to connect a subclass to its superclass.
- A sub-class is inherited from its superclass.
- A solid line with a hollow arrowhead that point from the child to the parent class.





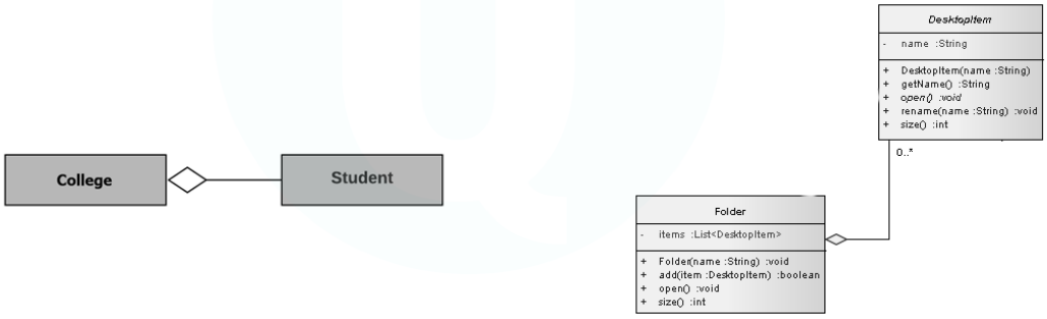
3)Association

- This kind of relationship represents static relationships between classes A and B.
- There is an association between Class1 and Class2
- A solid line connecting two classes



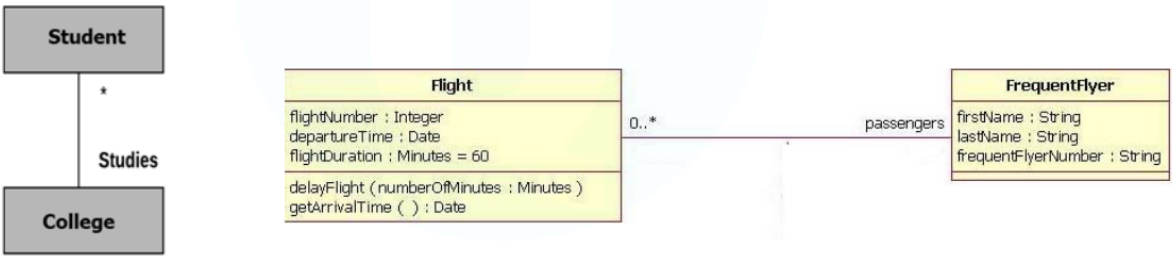
4)Aggregation

- A special type of association. It represents a "part of" relationship
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- A solid line with an unfilled diamond at the association end connected to the class of composite



5)Multiplicity

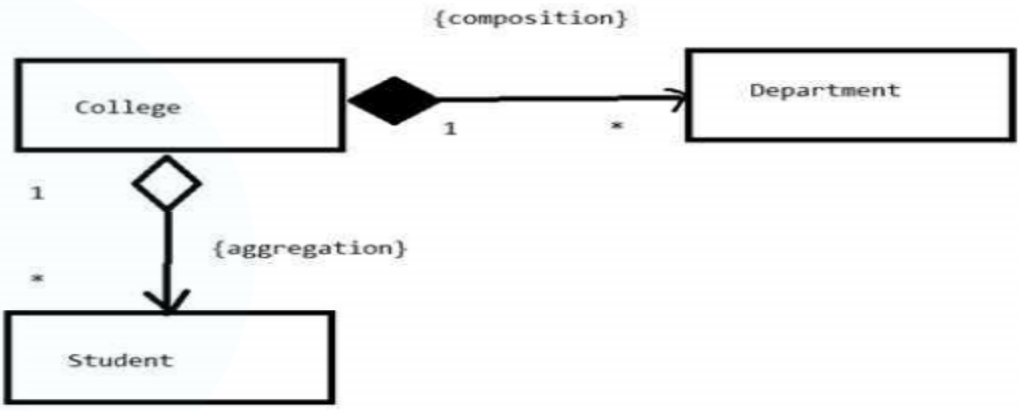
- This kind of relationship represents static relationships between classes A and B. Eg: A Person lives at an location.
- It should be named to indicate the role played by the class attached at the end of the association path.
- It means, how many objects of each class take part in the relationships
    - Exactly one - 1
    - Zero or one - 0..1
    - Many - 0..* or *
    - One or more - 1..*
    - Exact Number - e.g. 3..4 or 6
    - Or a complex relationship - e.g. 0..1, 3..4, 6.* would mean any number of objects other than 2 or 5



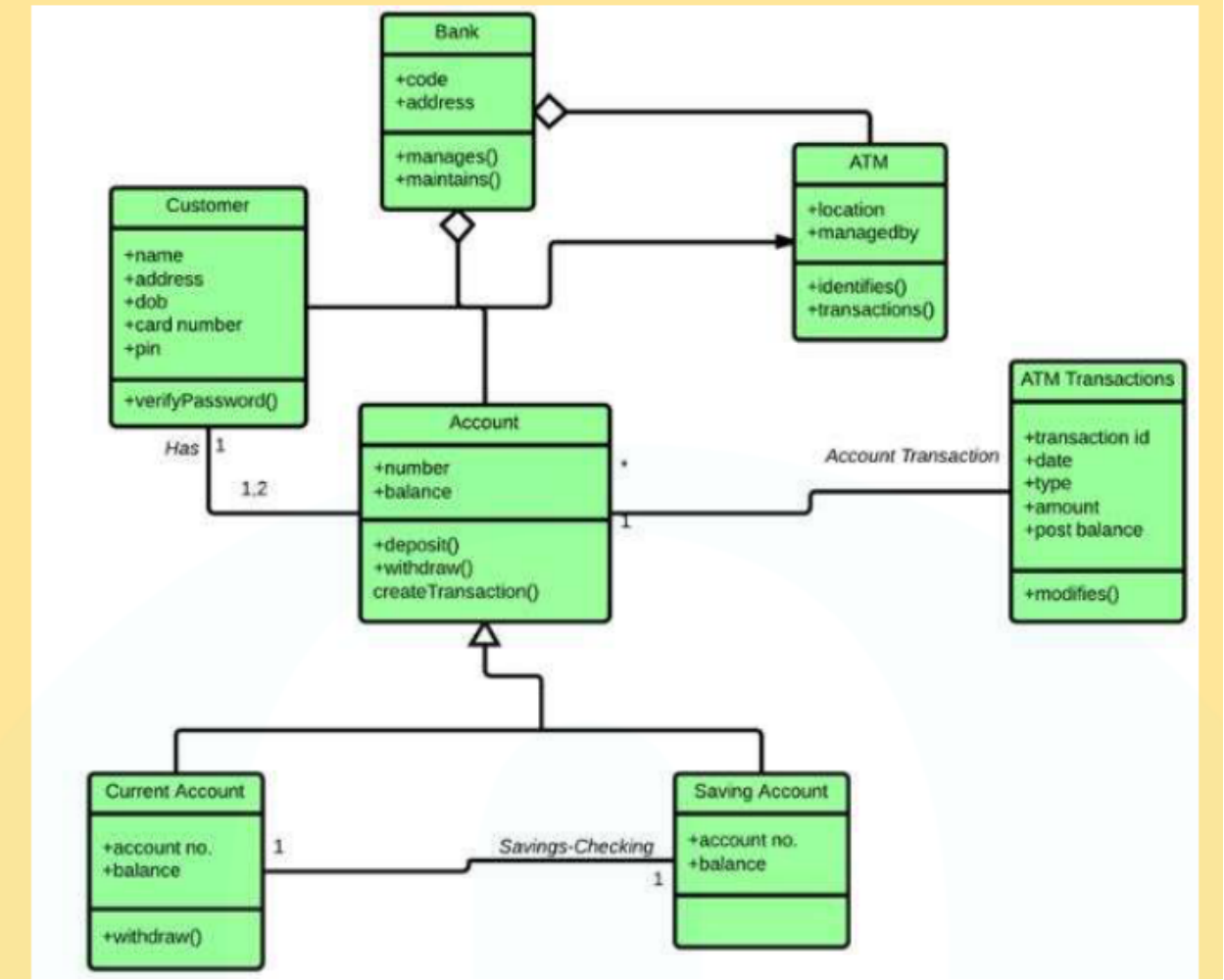
6)Composition

- A special type of aggregation where parts are destroyed when the whole is destroyed.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.

  • A solid line with a filled diamond at the association connected to the class of composite



Eg:- Class Diagram for ATM MAchine



2. USE CASE MODEL / USE CASE DIAGRAM

The purpose of a use case diagram in UML is to demonstrate the different ways that a user might interact with a system.

It captures the dynamic behavior of a live system.

A use case diagram can summarize the details of your **system's users** (also known as **actors**) and their interactions with the system.

To build a use case diagram, we will use a set of specialized symbols and connectors.

A use case diagram doesn't go into a lot of detail, but it depicts a high-level overview of the relationship between **use cases, actors, and systems.**

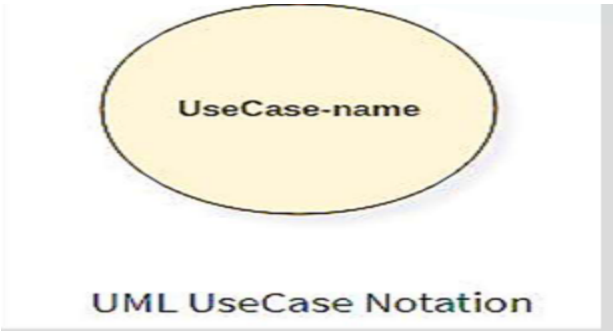
A use-case model is a model of how different types of users interact with the system to solve a problem

Use case diagram components:-

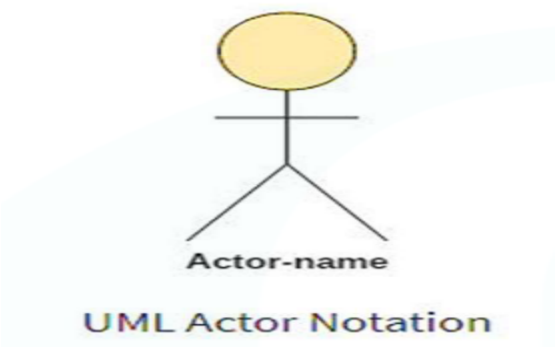
1) **Actors**:- The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
2) **System:-** A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a **scenario.**
3) **Goals:-** The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal

Use case diagram symbols and notation:-

1) Use cases:- Horizontally shaped ovals that represent the different uses that a user might have. A use case represents a distinct functionality of a system, a component, a package, or a class.



2) **Actors:-** Stick figures that represent the people actually employing the use cases. A user is the best example of an actor. One actor can be associated with multiple use cases in the system



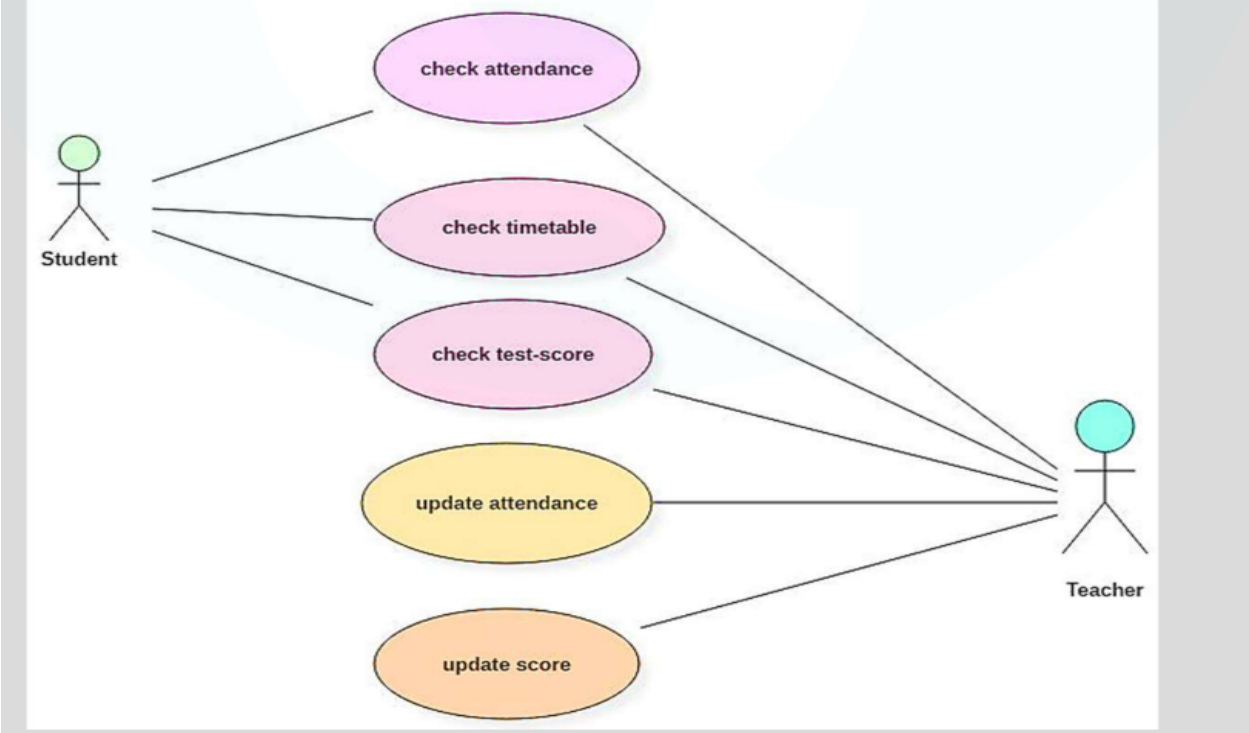
3) **Associations**:- A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
4) **System boundary boxes**:- A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system.

5) **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.

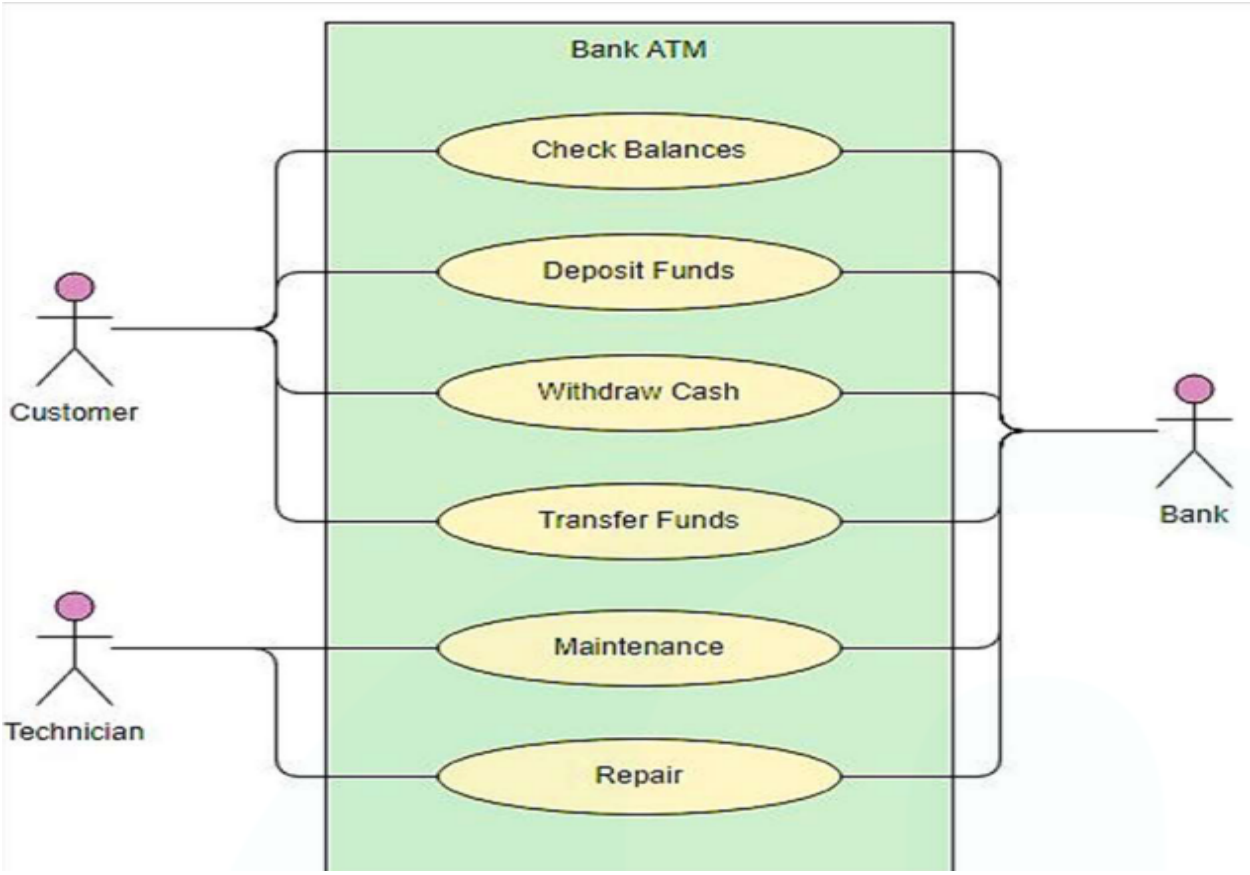
Purposes of use case diagram:-

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements and actors

Eg: Use case diagram of a student management system

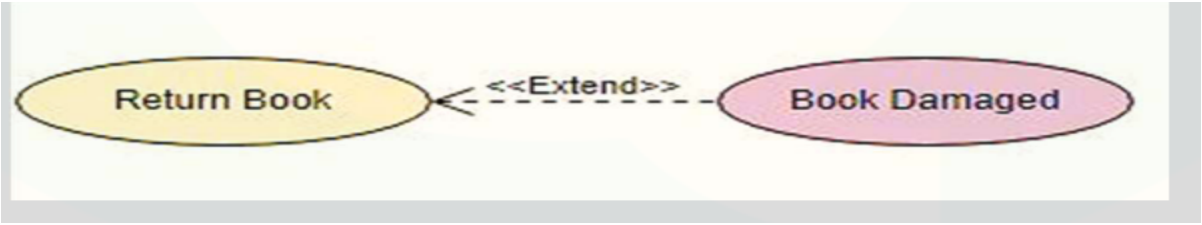


Eg: ATM use case diagram



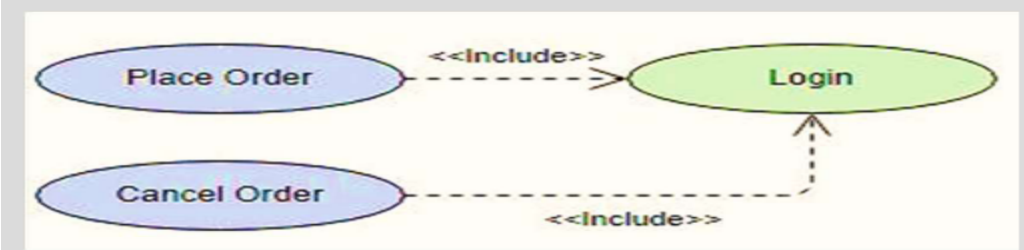
<<extend>> Use Case

The <<extend>> use case inserting additional action sequences into the base use-case sequence.

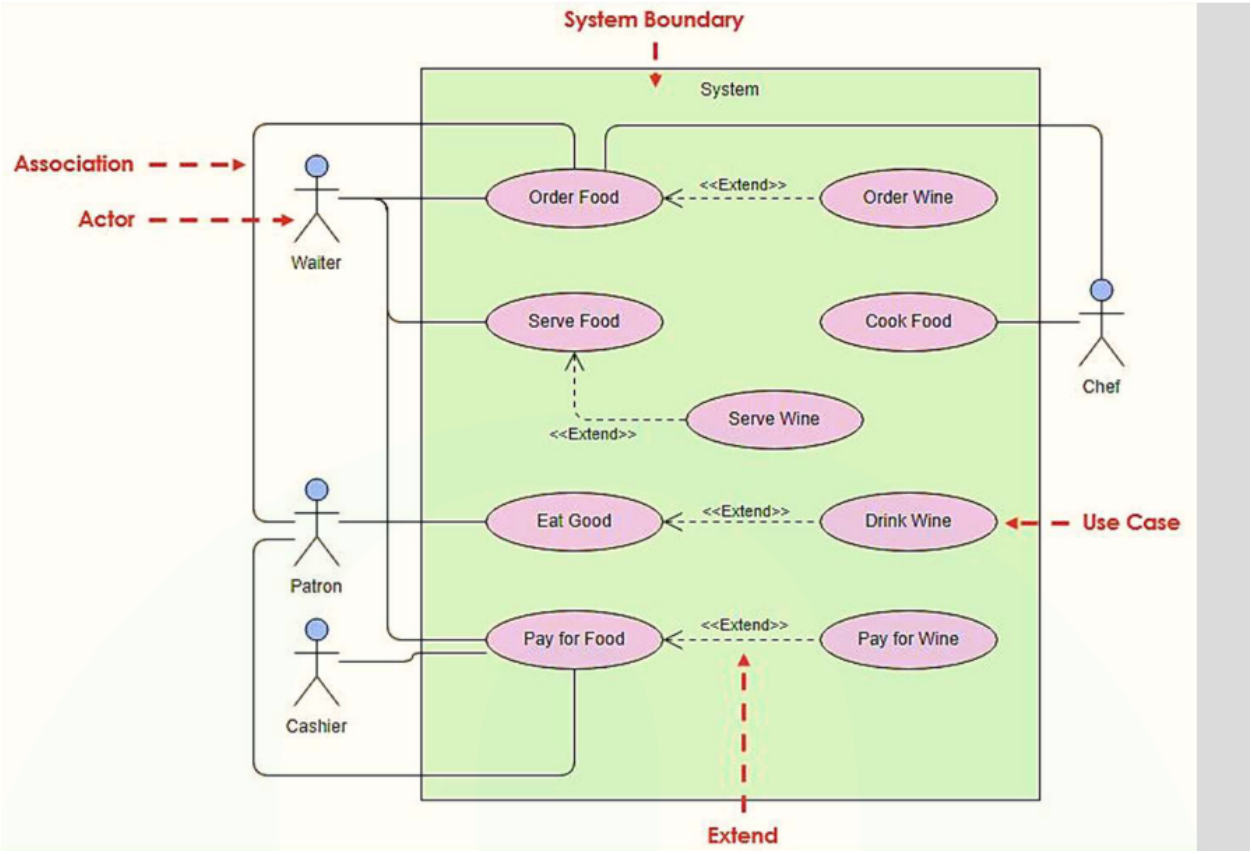


<<include>> Use Case

The time to use the <<include>> relationship is after you have completed the first cut description of all your main Use Cases.



Eg: Food and Drink ordering Use case Diagram



3. INTERACTION DIAGRAM

INTERACTION DIAGRAMS are used in UML to establish **communication between objects.**

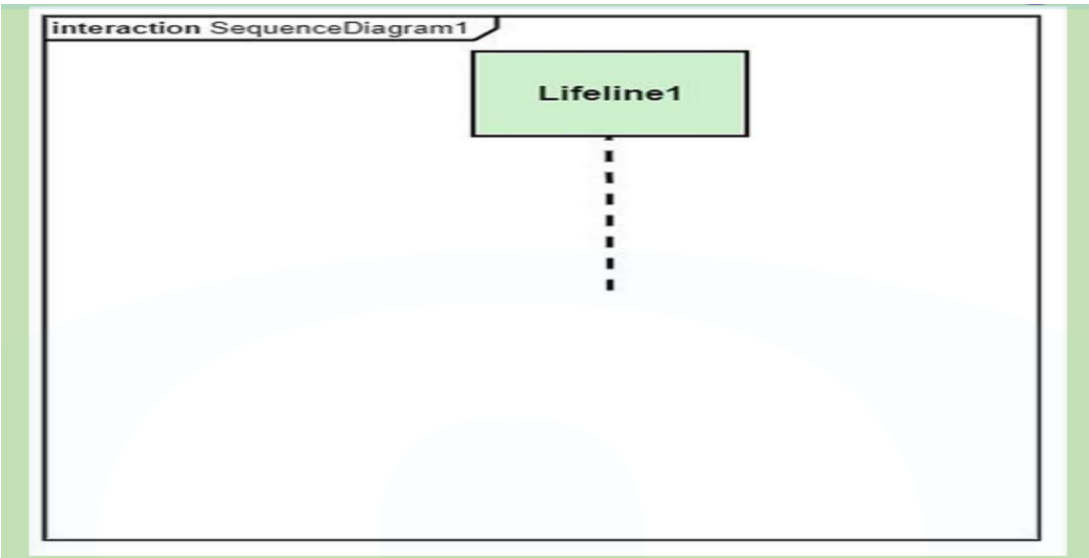
Interaction diagrams mostly focus on message passing and how these messages make up one functionality of a system.

The critical component in an interaction diagram is **lifeline and messages.**

Interaction diagrams capture the **dynamic behavior of any system.**

The details of interaction can be shown using several notations such as **sequence diagram, timing diagram, collaboration diagram.**

Notation of an Interaction Diagram:-



Purpose of an Interaction Diagram:-

- To capture the **dynamic behavior** of a system.
- To describe the **message flow** in the system.
- To describe the structural organization of the objects.
- To describe the **interaction among objects.**
- Interaction diagram **visualizes the communication and sequence of message passing** in the system.
- Interaction diagram represents the **ordered sequence of interactions** within a system.
- Interaction diagrams can be used to explain the **architecture of an object-oriented system.**

Different types of Interaction Diagrams:-

1. Sequence diagram

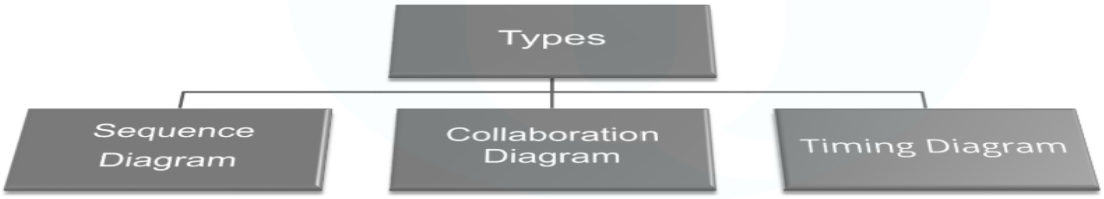
- Purpose - To visualize the sequence of a message flow in the system
- Shows the interaction between two lifelines

2. Collaboration diagram

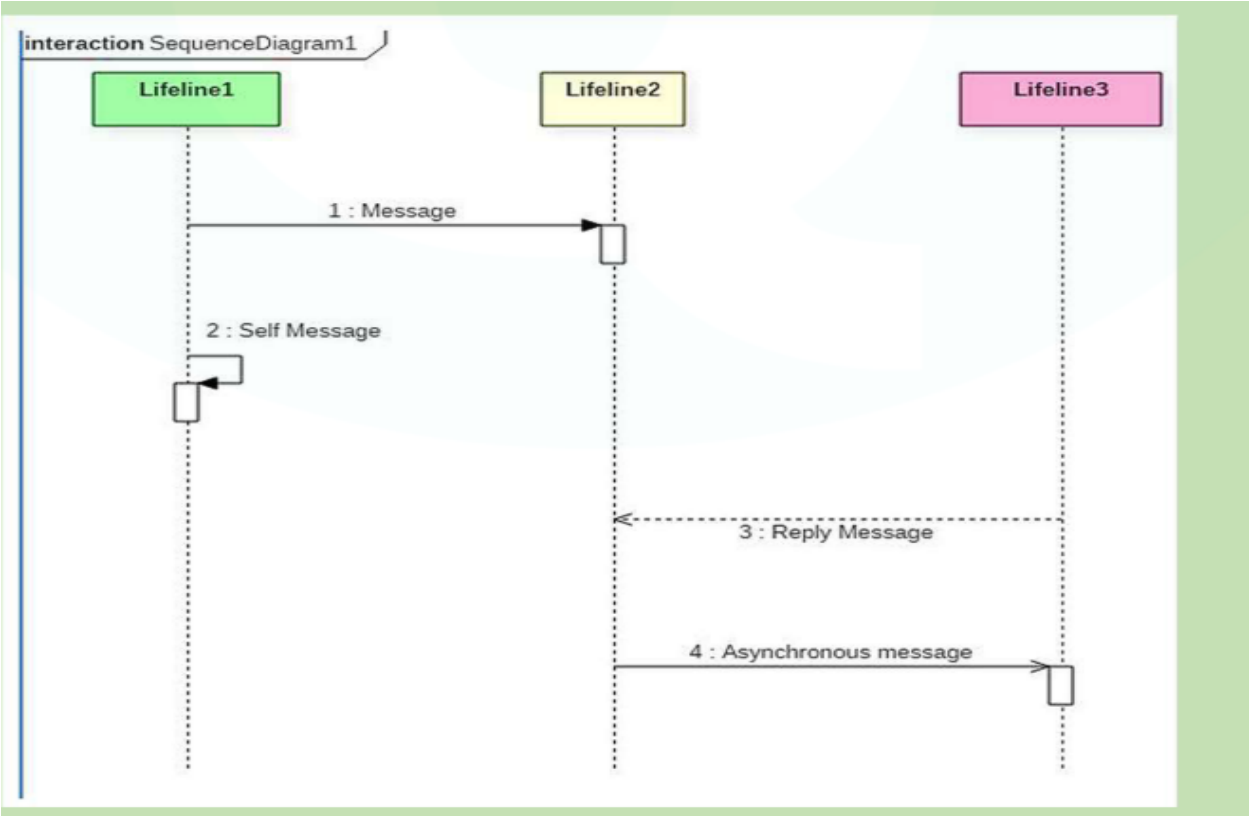
- Also called as a **communication diagram**
- Shows how various lifelines in the system connect**.**

3. Timing diagram

- Focus on the instance at which a message is sent from one object to another object.



How to draw a Sequence Diagram:-



In a sequence diagram, a **lifeline** is represented by a vertical bar.

A **lifeline** represents an individual participant in a sequence diagram.

A **lifeline** will usually have a rectangle containing its object name.

A message flow between two or more objects is represented using a vertical dotted line which extends across the bottom of the page.

In a sequence diagram, different types of messages and operators are used.

In a sequence diagram, iteration and branching are also used.

3.1) Sequence Diagram

A Sequence Diagram simply depicts **interaction between objects in a sequential order.**

The purpose of a sequence diagram in UML is to visualize the sequence of a message flow in the system.

Messages – Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline.

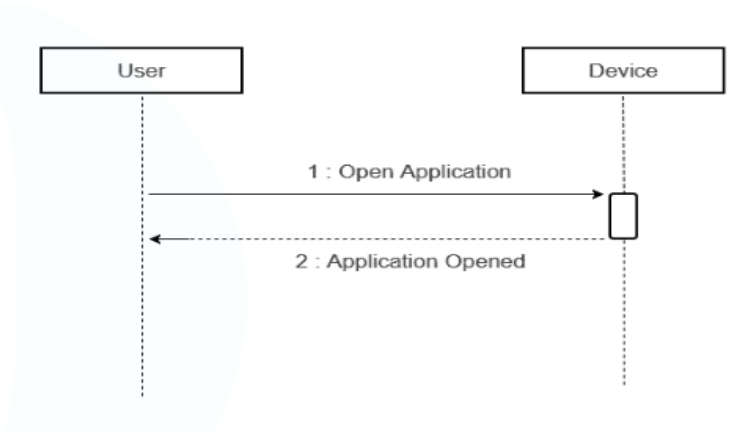
Sequence Diagram Messages:-

1. **Synchronous message -**

   A synchronous message waits for a reply before the interaction can move forward.

   The sender waits until the receiver has completed the processing of the message.

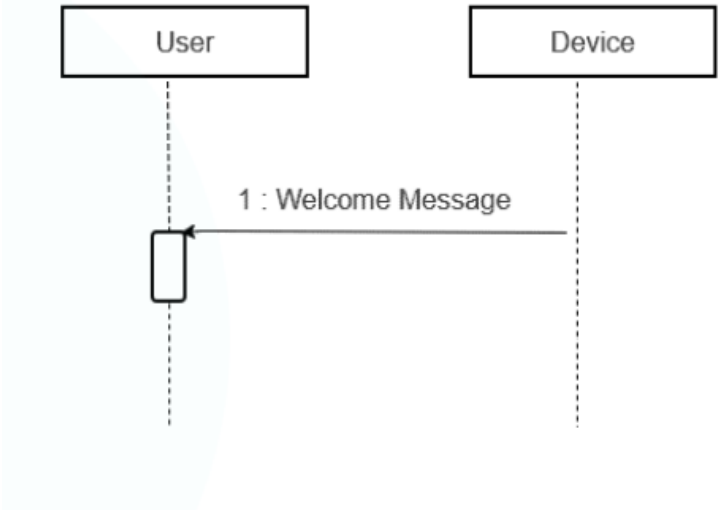
   The caller continues only when it knows that the receiver has processed the previous message.



2. **Asynchronous message -**

   An asynchronous message does not wait for a reply from the receiver.

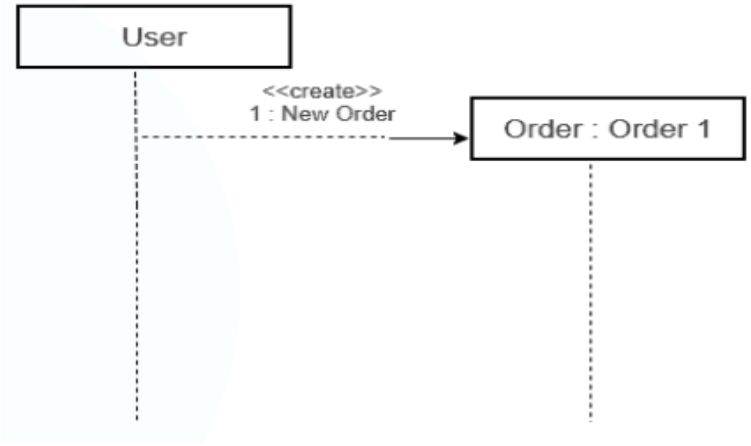
   The interaction moves forward irrespective of the receiver processing the previous message or not.



3. **Create message -**

   We use a Create message to instantiate a new object in the sequence diagram.

   There are situations when a particular message call requires the creation of an object.

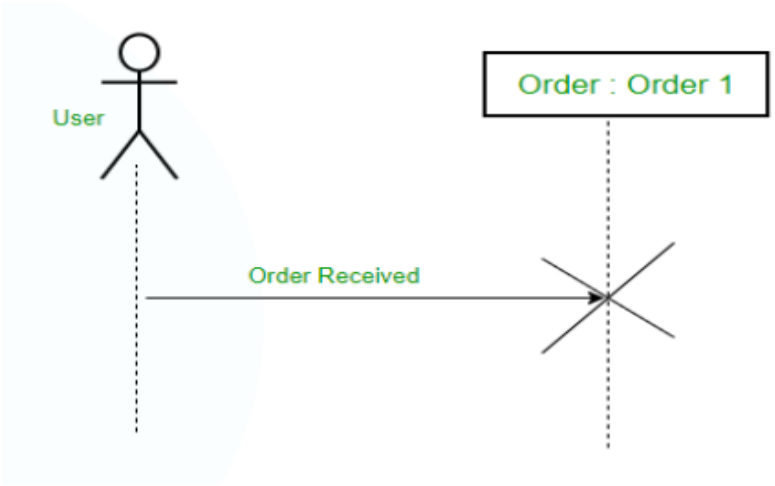


4. **Delete Message -**

   We use a Delete Message to delete an object.

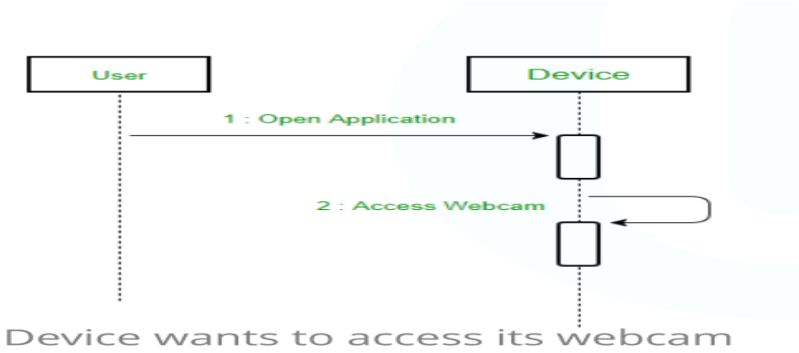
   When an object is deallocated memory or is destroyed within the system we use the Delete Message symbol.

   It destroys the occurrence of the object in the system.



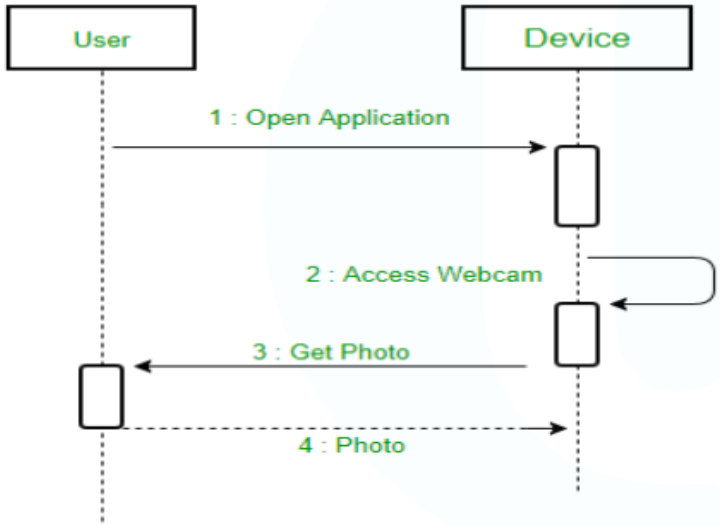
5. **Self Message -**

   Certain scenarios might arise where the object needs to send a message to itself.



6. Reply Message -

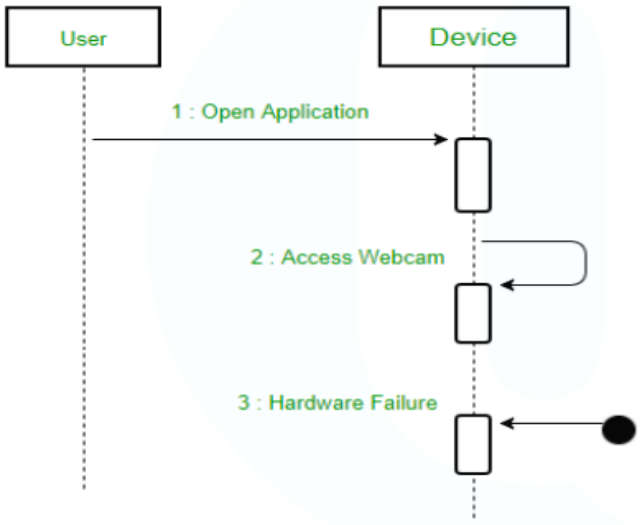
Reply messages are used to show the message being sent from the receiver to the sender.



A scenario where a reply message is used

6. **A Found message**

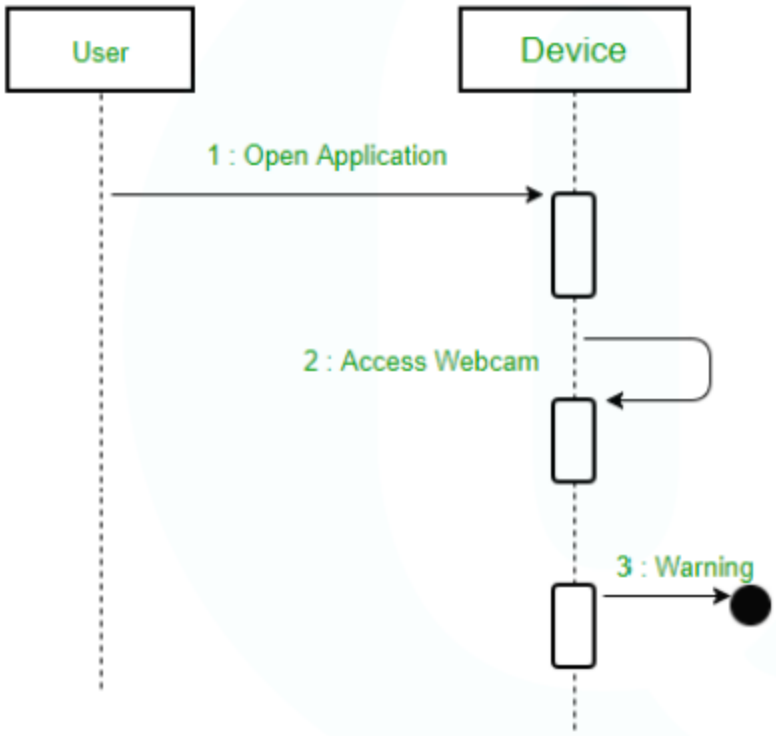
   A Found message is used to represent a scenario where an unknown source sends the message.



A scenario where a found message is used

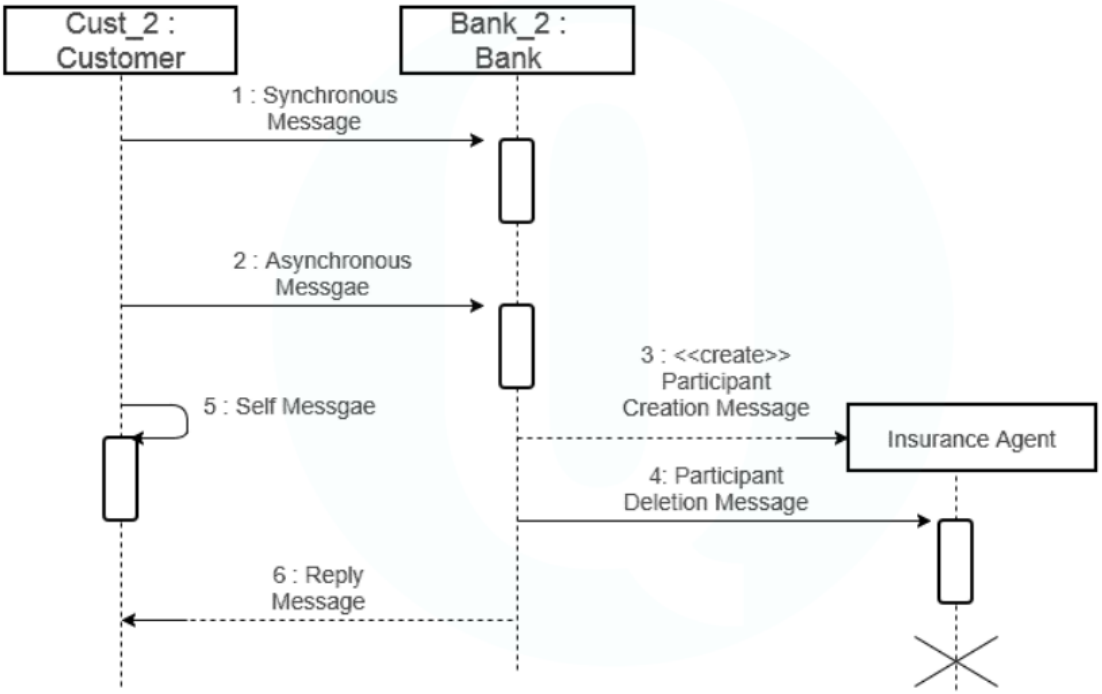
7. **Lost message**

   A Lost message is used to represent a scenario where the recipient is not known to the system.

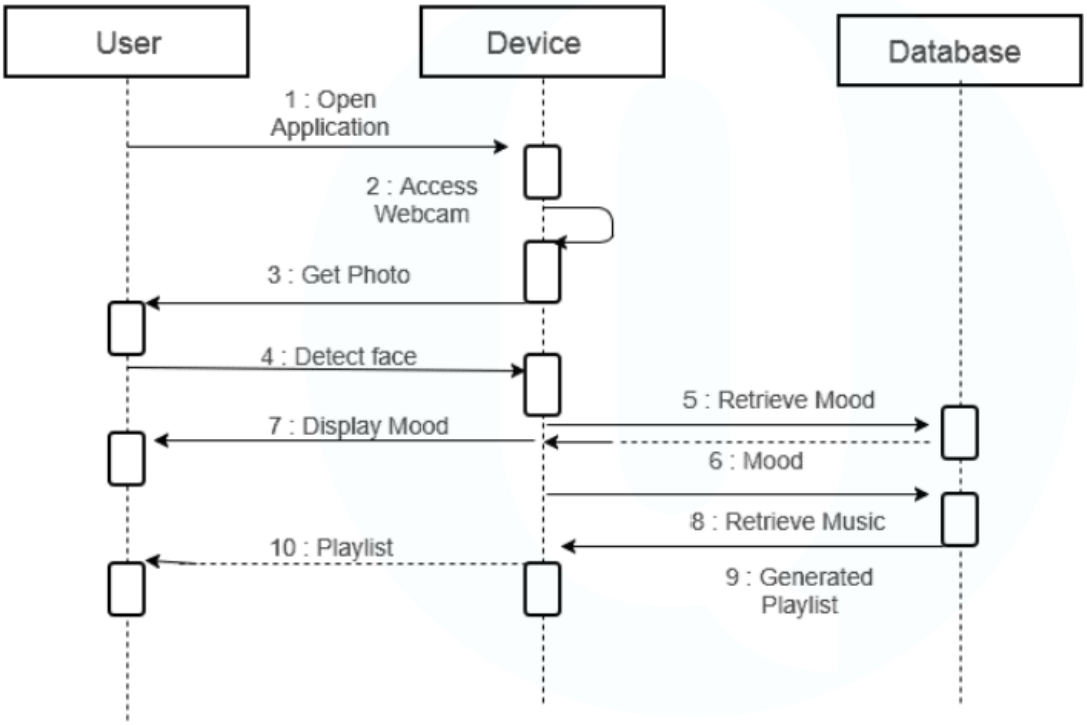


scenario where a lost message is used

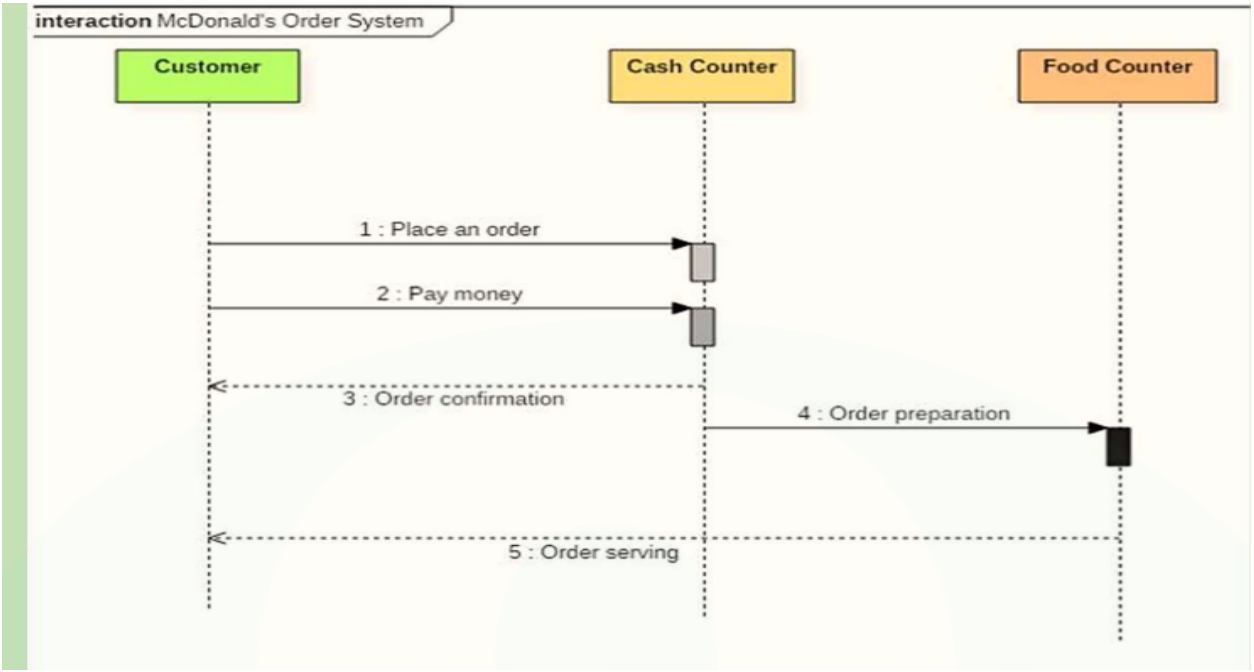
Sequence Diagram - Messages:-



Eg: Sequence Diagram for Emotion Based Music Player



Eg: Sequence Diagram for McDonald Food Ordering System

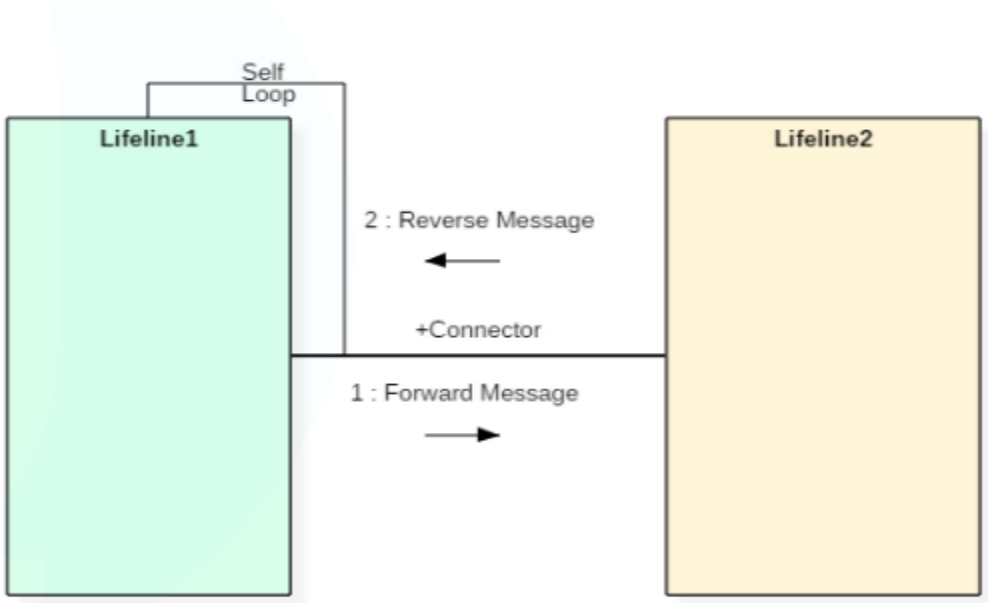


Benefits of Sequence Diagram:-

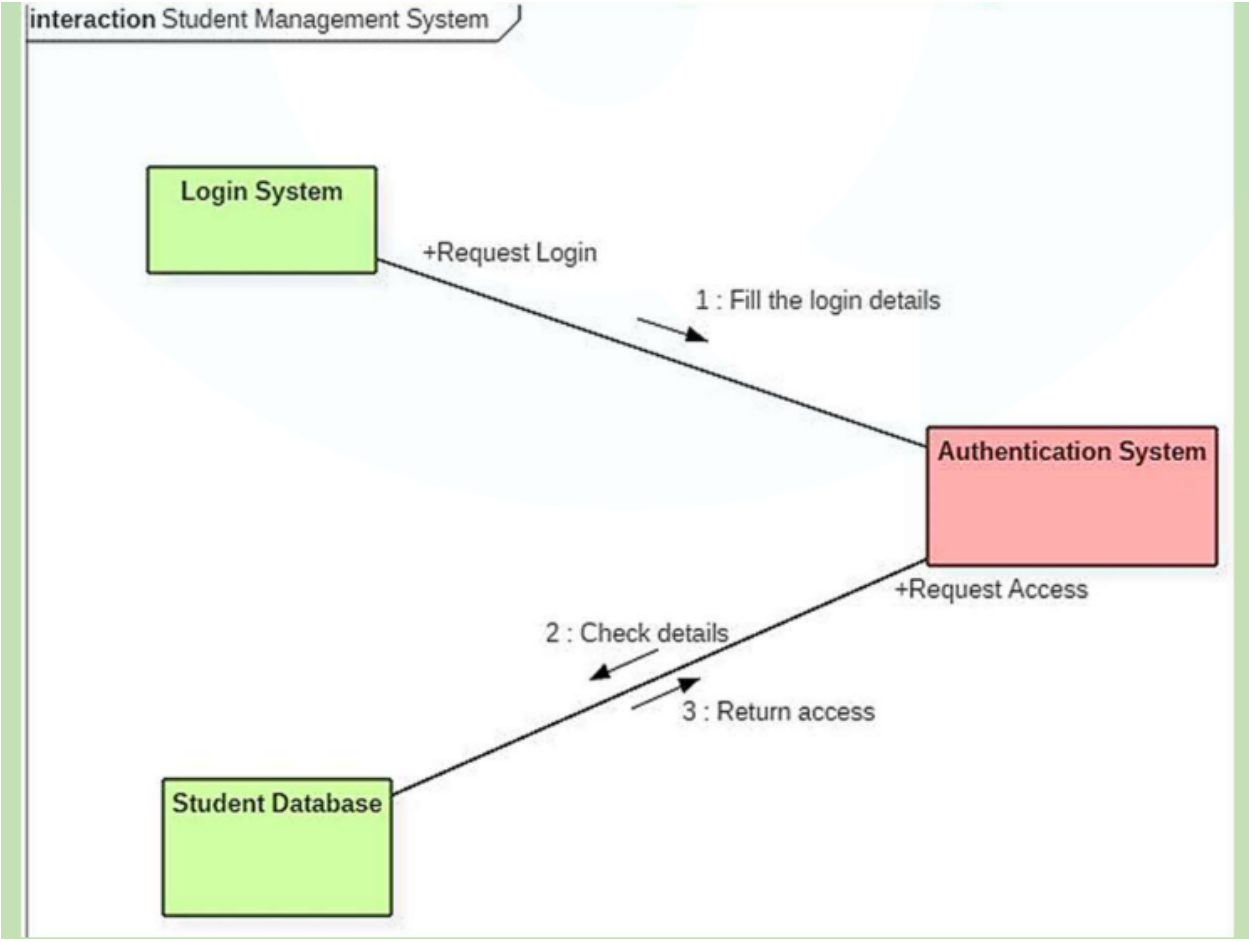
- Used to explore any real application or a system.
- Used to represent message flow from one object to another object.
- Easier to maintain and generate.
- Can be easily updated according to the changes within a system.
- Allows reverse as well as forward engineering.

3.2) Collaboration Diagram

Collaboration represents the **relationships and interactions among software objects.**
They are used to understand the **object architecture within a system** rather than the flow of a message as in a sequence diagram.



Eg: Collaboration Diagram for Student Management System:-



The above collaboration diagram represents a student information management system. The flow of communication in the above diagram is given by, A student requests a login through the login system. An authentication mechanism of software checks the request. If a student entry exists in the database, then the access is allowed; otherwise, an error is returned.

Benefits of Collaboration Diagram:-

- It is also called a **communication diagram.**
- It emphasizes the structural aspects of an interaction diagram - how lifeline connects.
- Its syntax is similar to that of sequence diagram except that lifeline don't have tails.
- Messages passed over sequencing is indicated by numbering each message hierarchically.
- Compared to the sequence diagram communication diagram is semantically weak.
- Object diagrams are special case of communication diagram.
- It allows you to focus on the elements rather than focusing on the message flow as described in the sequence diagram.
- Sequence diagrams can be easily converted into a collaboration diagram as collaboration diagrams are not very expressive.

- While modeling collaboration diagrams w.r.t sequence diagrams, some information may be lost.

Drawbacks of Collaboration Diagram:-

- Collaboration diagrams can become complex when too many objects are present within the system.
- It is hard to explore each object inside the system.
- Collaboration diagrams are time consuming.
- The object is destroyed after the termination of a program.
- The state of an object changes momentarily, which makes it difficult to keep track of every single change the occurs within an object of a system.

3.3) Timing diagram

Timing diagram is a waveform or a graph that is used to describe the state of a lifeline at any instance of time.

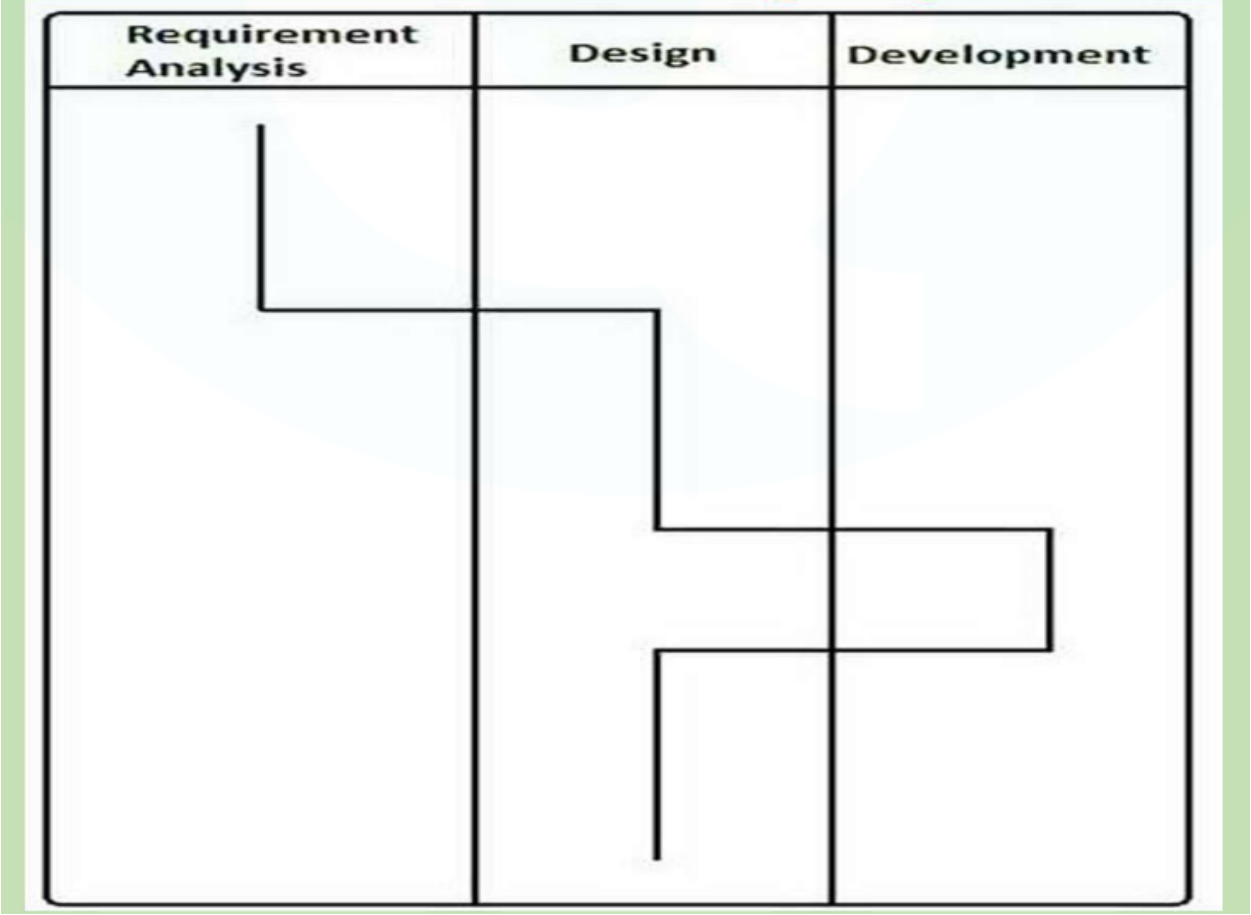
It is used to denote the transformation of an object from one form into another form.

Timing diagram does not contain notations as required in the sequence and collaboration diagram.

The flow between the software program at various instances of time is represented using a waveform.

How to draw a Timing Diagram:-

In the below diagram, first, the software passes through the requirements phase then the design and later the development phase. The output of the previous phase at that given instance of time is given to the second phase as an input Thus, the timing diagram can be used to describe SDLC (SoftwareDevelopment Life Cycle) in UML.



Benefits of a Timing Diagram:-

• Timing diagrams are used to represent the state of an object at a particular instance of time.
• Timing diagram allows reverse as well as forward engineering.
• Timing diagrams can be used to keep track of every change inside the system.

Drawbacks of a Timing Diagram

• Timing diagrams are difficult to understand.
• Timing diagrams are difficult to maintain

4) ACTIVITY DIAGRAM

ACTIVITY DIAGRAM is basically a flowchart to represent the **flow from one activity to another activity.**

The **activity** can be described as an **operation of the system.**

The basic purpose of activity diagrams is to capture the dynamic behavior of the system.

It is also called **object-oriented flowchart.**

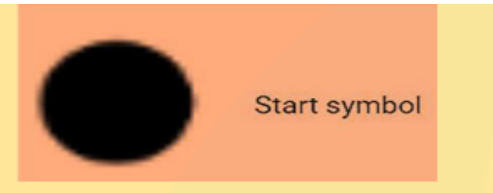
Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques.

Basic components of an activity diagram:-

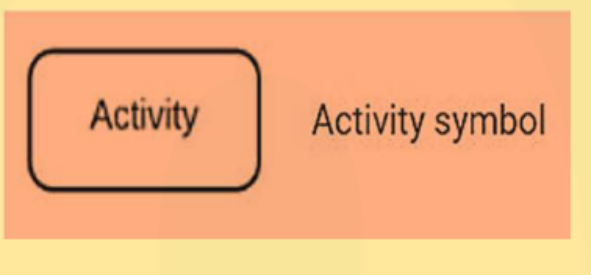
- **Action:** A step in the activity wherein the users or software perform a given task.
- **Decision node**: A conditional branch in the flow that is represented by a diamond. It includes a single input and two or more outputs.
- **Control flows:** Another name for the connectors that show the flow between steps in the diagram.
- **Start node:** Symbolizes the beginning of the activity. The start node is represented by a black circle.
- **End node:** Represents the final step in the activity. The end node is represented by an outlined black circle.

Activity diagram symbols:-

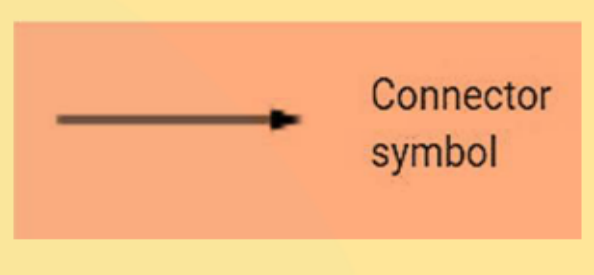
- **Start symbol -** Represents the beginning of a process or workflow in an activity diagram.



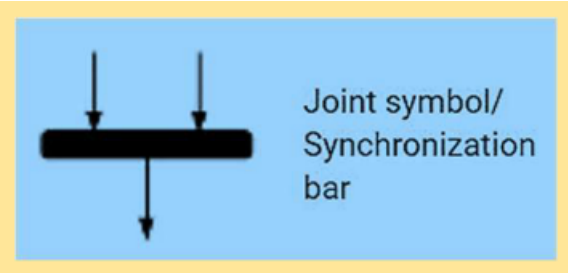
- **Activity symbol** - Indicates the activities that make up a modeled process. These symbols, which include short descriptions within the shape, are the main building blocks of an activity diagram.



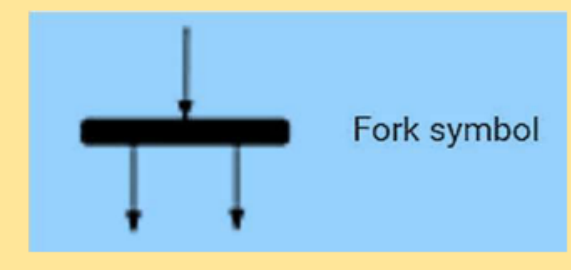
- **Connector symbol** - Shows the directional flow, or control flow, of the activity.



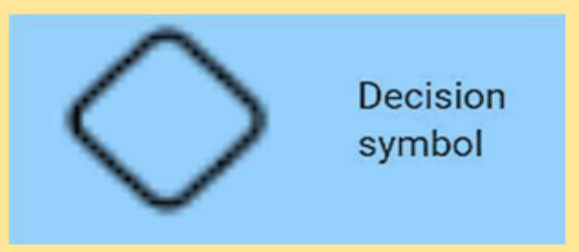
- **Joint symbol / Synchronization bar** - Combines two concurrent activities and re-introduces them to a flow where only one activity occurs at a time. Represented with a thick vertical or horizontal line.



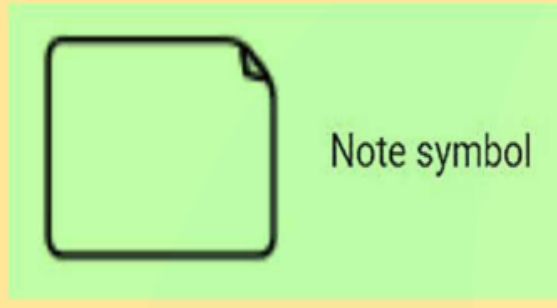
- **Fork symbol -** Splits a single activity flow into two concurrent activities. Symbolized with multiple arrowed lines from a join.



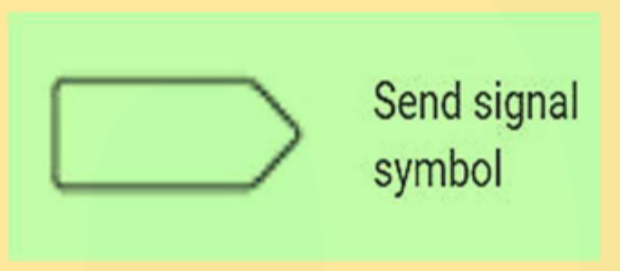
- **Decision symbol** - Represents a decision and always has at least two paths branching out with condition text.



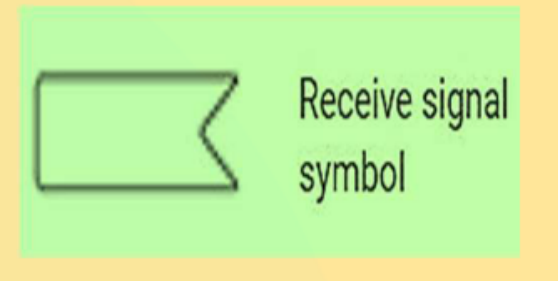
- **Note symbol -** Allows the diagram creators or collaborators to communicate additional messages that don't fit within the diagram itself. Leave notes for added clarity and specification.



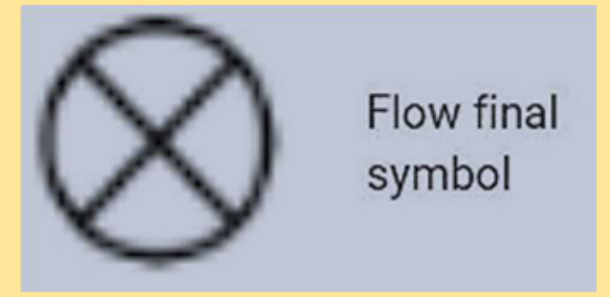
- **Send signal symbol** - Indicates that a signal is being sent to a receiving activity.



- **Receive signal symbol** - Demonstrates the acceptance of an event. After the event is received, the flow that comes from this action is completed.

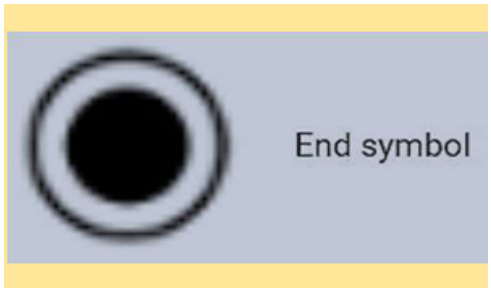


- **Flow final symbol** - Represents the end of a specific process flow. This symbol shouldn't represent the end of all flows in an activity. The flow final symbol should be placed at the end of a single activity flow.

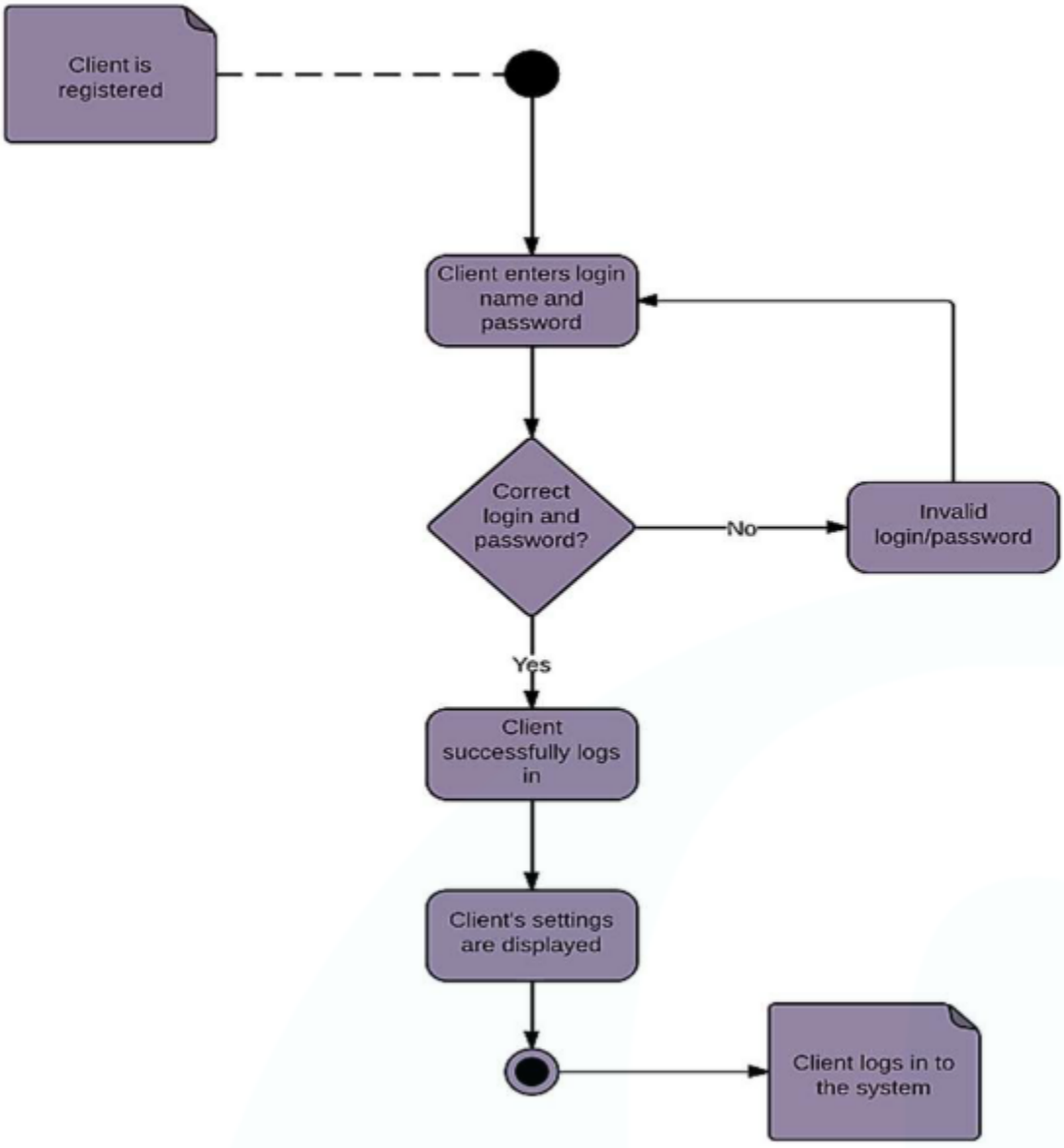


- **Condition text** - Placed next to a decision marker to let you know under what condition an activity flow should split off in that direction.

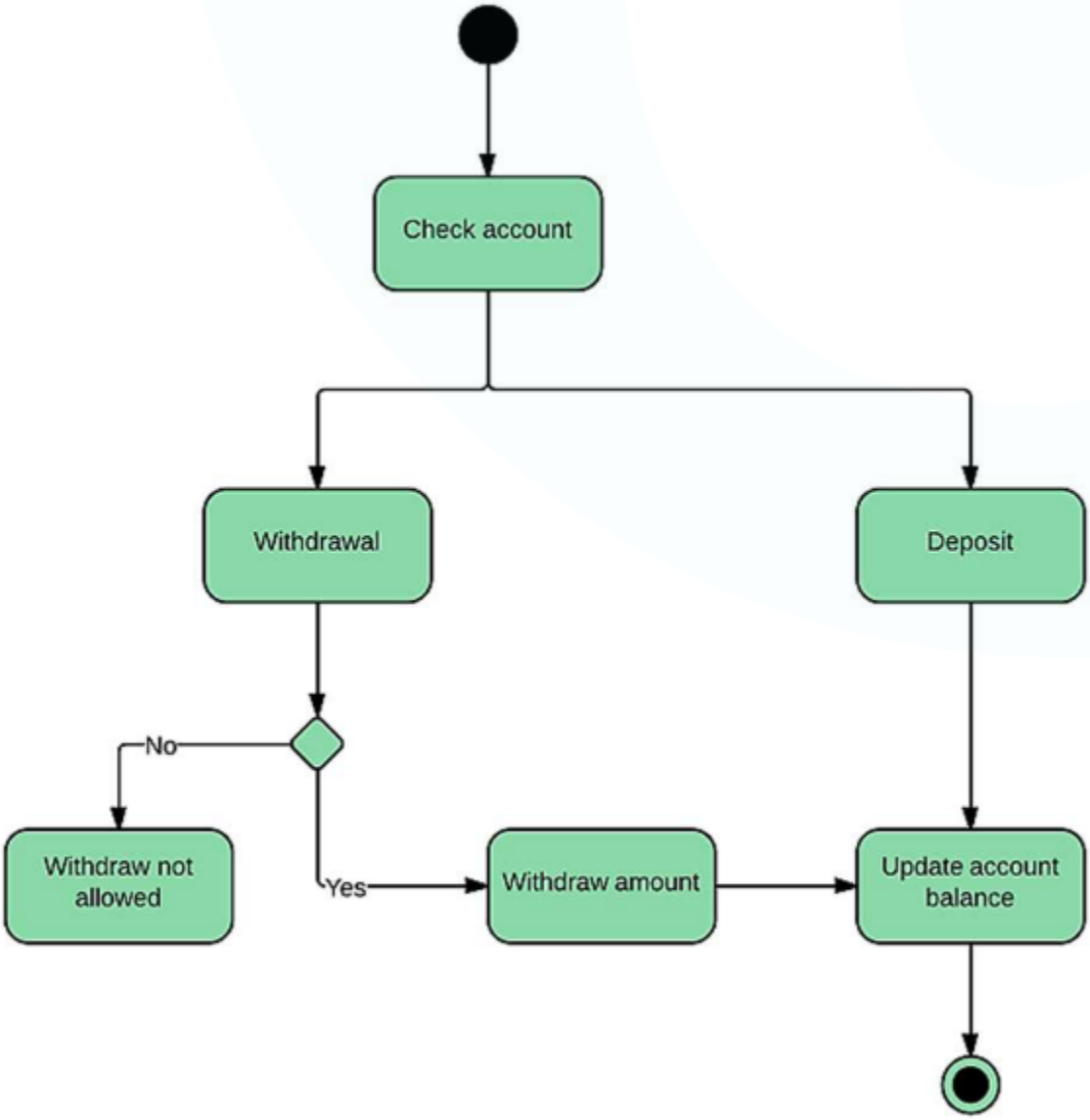
- **End symbol** - Marks the end state of an activity and represents the completion of all flows of a process.



Eg: Activity diagram - a login page



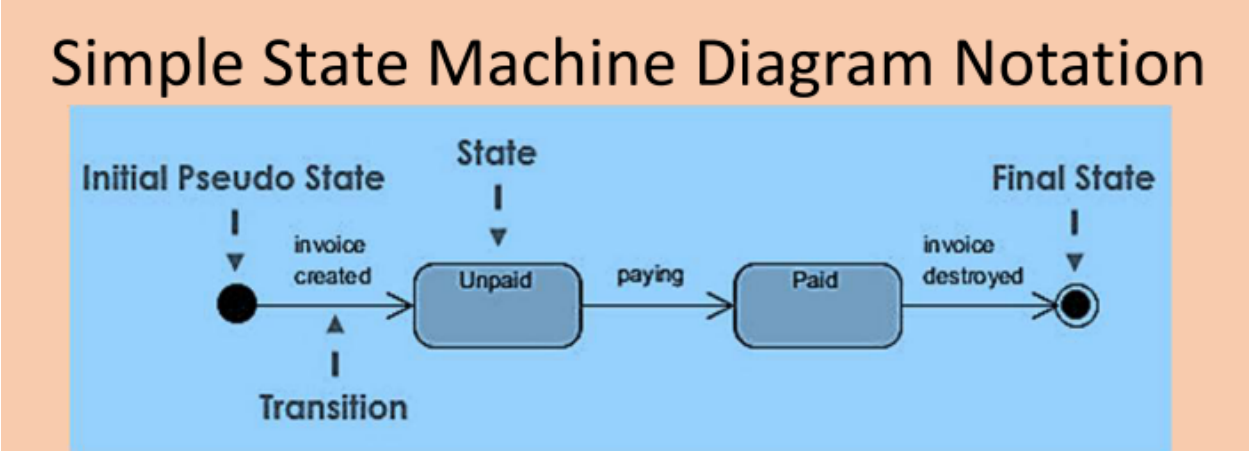
Eg:Activity Diagram - Banking system



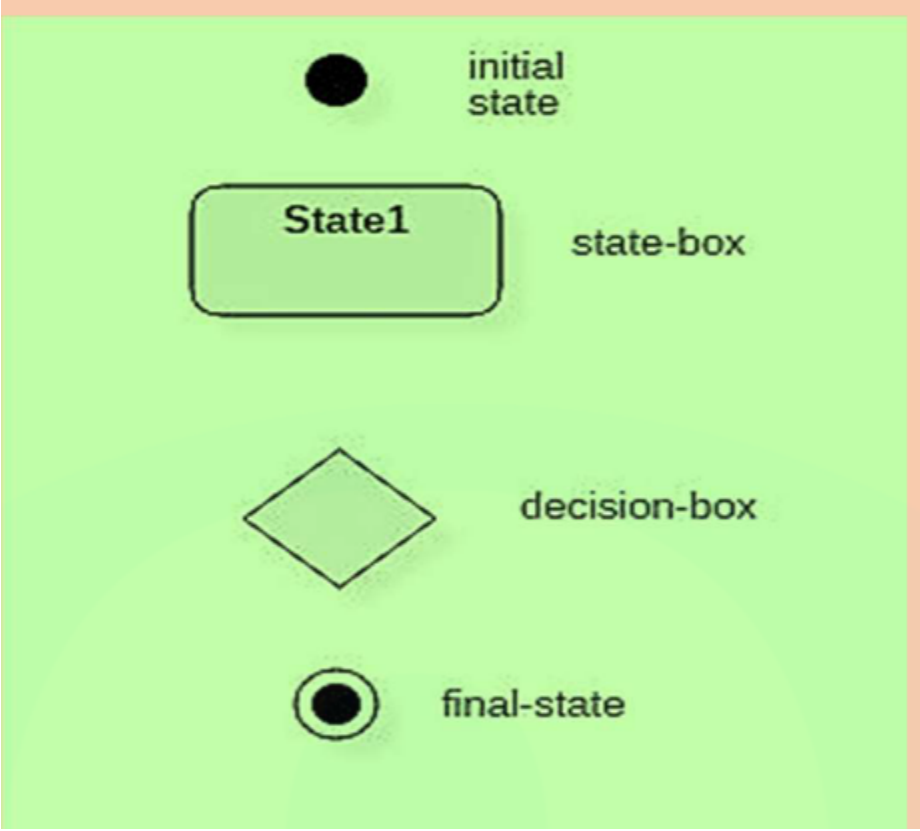
5) STATE CHART DIAGRAM

State chart diagram is used to capture the **dynamic aspect of a system.**
An object goes through various states during its lifespan.
The lifespan of an object remains until the program is terminated.
The object goes from multiple states depending upon the event that occurs within the object.
Each state represents some unique information about the object.
State chart diagram visualizes the **flow of execution from one state to another state of an object.**
It represents **the state of an object** from the creation of an object until the object is destroyed or terminated.

The primary purpose of a state chart diagram is to model interactive systems and **define each and every state of an object.**
State chart diagrams are also referred to as State machines and state diagrams.
A **state** machine consists of **states, linked by transitions**.
A state is a condition of an object in which it performs some activity or waits for an event



Notation and Symbol for State Machine:-



1. **Initial state -** The initial state symbol is used to indicate the beginning of a state machine diagram.

2. **Final state -** This symbol is used to indicate the end of a state machine diagram.
3. **Decision box -** It contains a condition. Depending upon the result of an evaluated guard condition, a new path is taken for program execution.
4. **Transition -** A transition is a change in one state into another state which is occurred because of some event. A transition causes a change in the state of an object.

State box:-

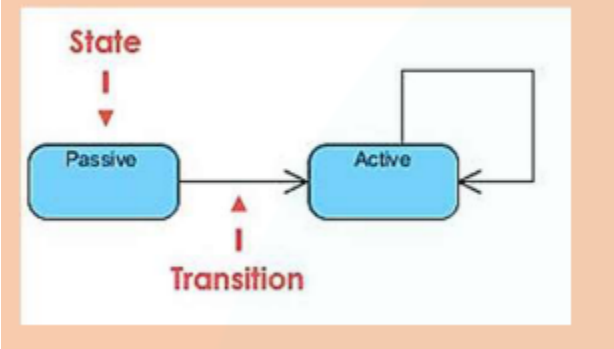
States represent situations during the **life of an object.**

It is denoted using a rectangle with round corners.

The name of a state is written inside the rounded rectangle.

A state can be either **active or inactive**.

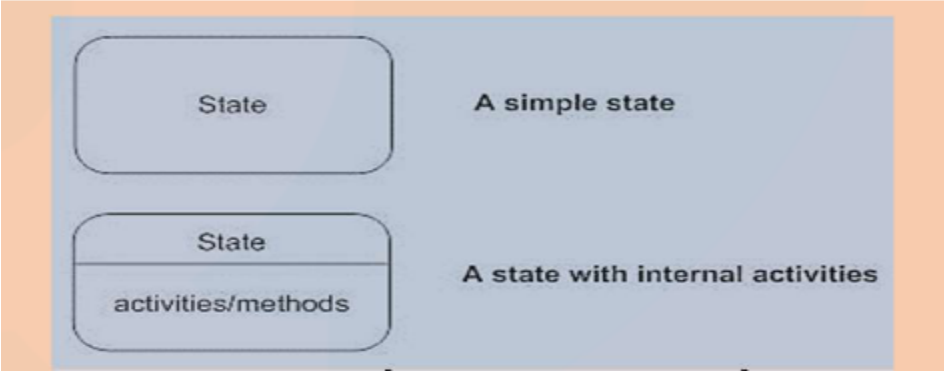
When a state is in the working mode, it is active, as soon as it stops executing and transits into another state, the previous state becomes inactive, and the current state becomes active.



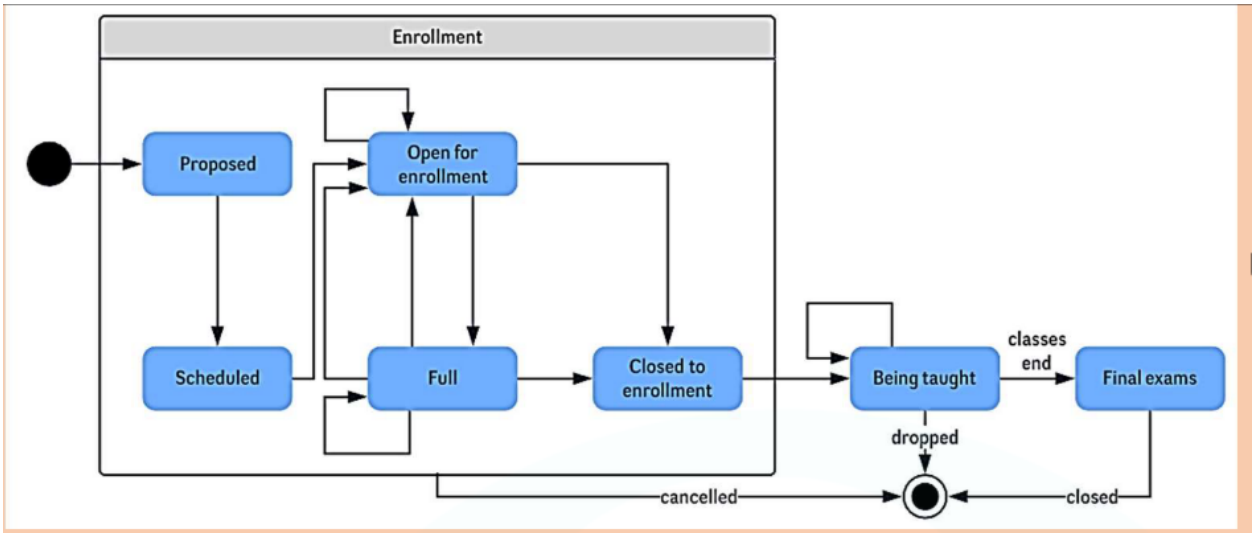
Eg: State chart Diagram for Online shopping

Types of State:-

1. **Simple state**
   They do not have any sub state.
2. **Composite state**
   These types of states can have one or more than one sub state.
   A composite state with two or more sub states is called an orthogonal state.
3. **Submachine state**
   These states are semantically equal to the composite states.
   Unlike the composite state, we can reuse the submachine states.

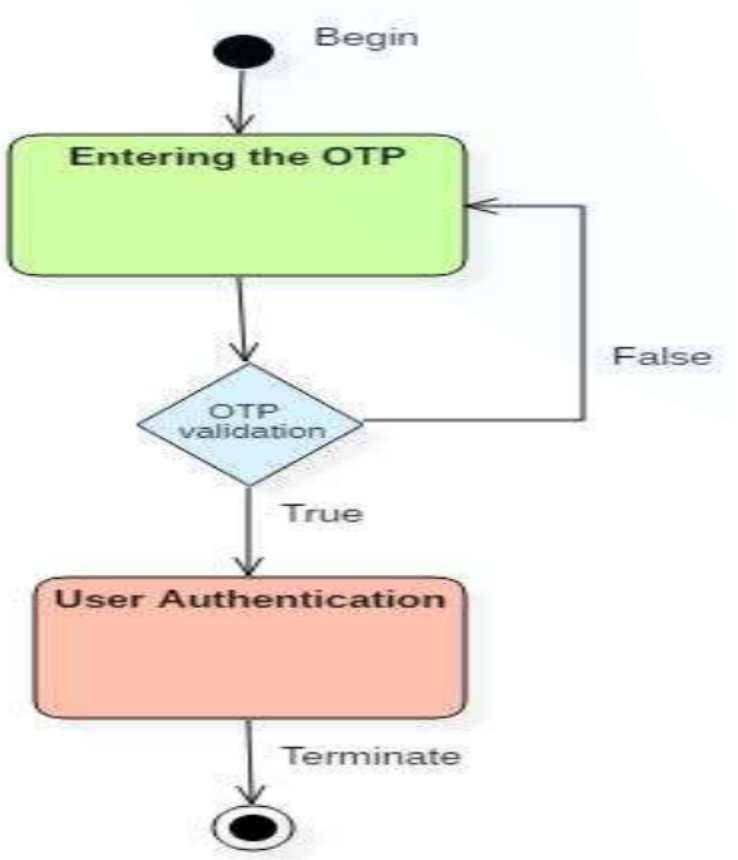


Eg : State chart Diagram for University Enrollment



The composite state “Enrollment” is made up of various sub states that will lead students through the enrollment process.
Once the student has enrolled, they will proceed to “Being taught” and finally to “Final exams.

Eg: State chart Diagram for User Authentication



State machine vs. Flowchart:

Statemachine	FlowChart
It represents various states of a system.	The Flowchart illustrates the program execution flow.
The state machine has a WAIT concept, i.e., wait for an action or an event.	The Flowchart does not deal with waiting for a concept.
State machines are used for a live running system.	Flowchart visualizes branching sequences of a system.
The state machine is a modeling diagram.	A flowchart is a sequence flow or a DFD diagram.
The state machine can explore various states of a system.	Flowchart deal with paths and control flow.