Unit 4

Behavioral Modeling

Use case Diagram Identify Actors

Identify Use cases: describing how the user will use the system Develop use-case Model

Description of Use case Diagram

Interaction Diagrams:

Activity Diagram

Sequence diagram

Collaboration Diagram

State Transition Diagram

Unified Modeling Language (UML) Diagrams

Unified Modeling Language (UML) is a general-purpose modeling language.

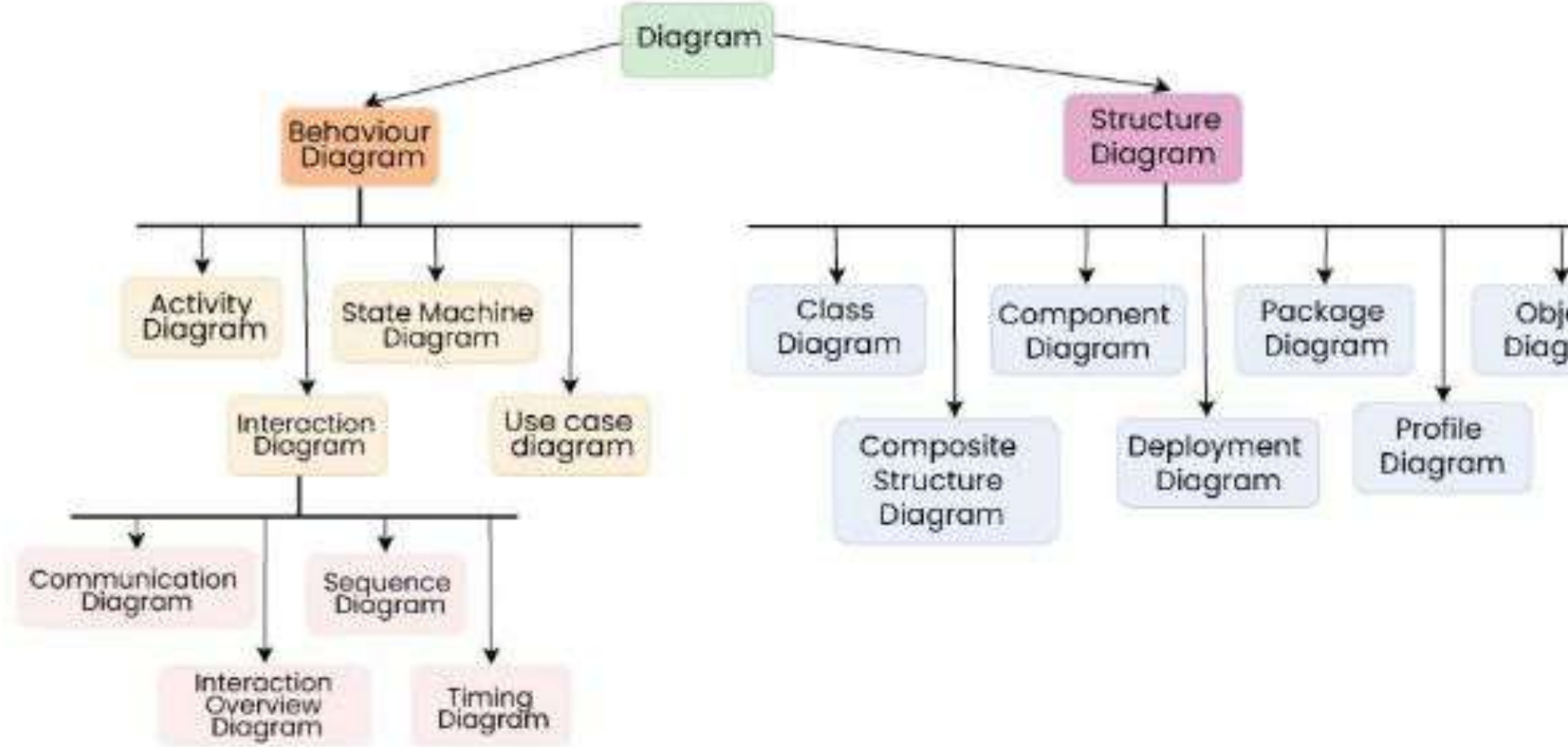
The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

UML is not a programming language, it is rather a visual language.

- We use UML diagrams to show the behavior and structure of a system.
- UML helps software engineers, businessmen, and system architects with modeling, design, and analysis.
- The International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.

Why do we need UML?

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code.
- So UML becomes essential to communicate with non-programmers about essential requirements, functionalities, and processes of the system.
- A lot of time is saved down the line when teams can visualize processes, user interactions, and the static structure of the system.



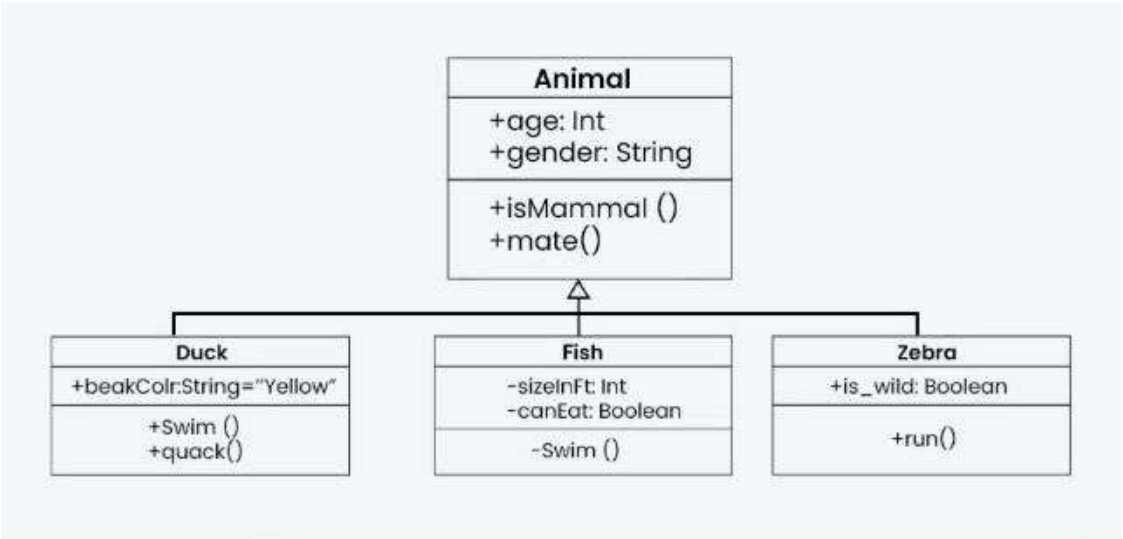
UML Diagrams

Structural UML Diagrams

- Structural UML diagrams are visual representations that depict the static aspects of a system, including its classes, objects, components, and their relationships, providing a clear view of the system's architecture.
- Structural UML diagrams include the following types:

Class Diagram

- The most widely use UML diagram is the class diagram.
- It is the building block of all object oriented software systems.
- We use class diagrams to depict the static structure of a system by showing system's classes, their methods and attributes.
- Class diagrams also help us identify relationship between different classes or objects.



Class Diagram example

Composite Structure Diagram

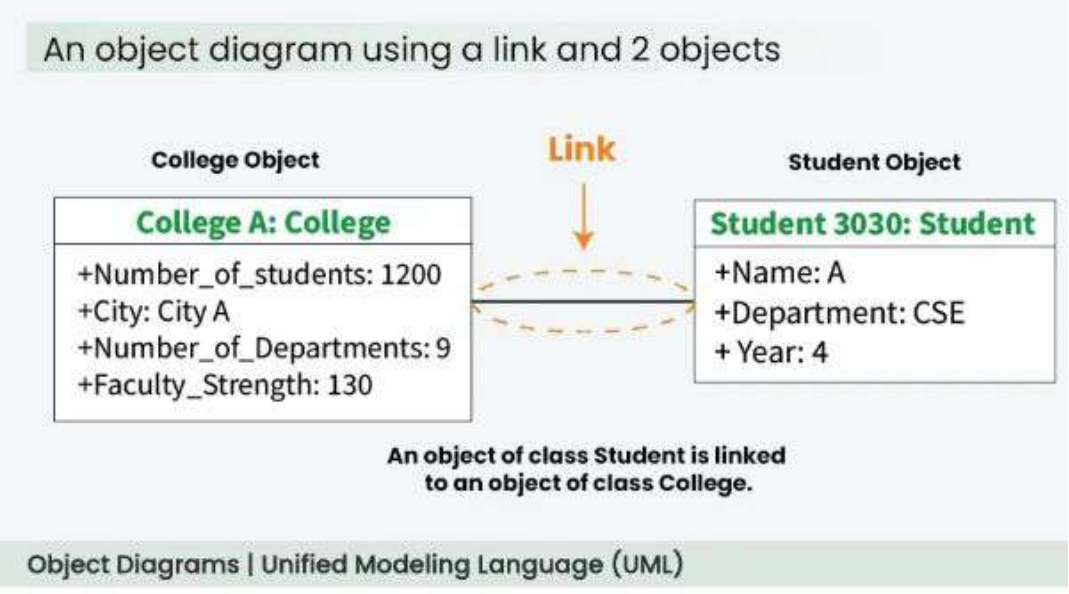
We use composite structure diagrams to represent the internal structure of a class and its interaction points with other parts of the system.

- A composite structure diagram represents relationship between parts and their configuration which determine how the classifier (class, a component, or a deployment node) behaves.
- They represent internal structure of a structured classifier making the use of parts, ports, and connectors.
- We can also model collaborations using composite structure diagrams.
- They are similar to class diagrams except they represent individual parts in detail as compared to the entire class.

Object Diagram

An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them. Since object diagrams depict behaviour when objects have been instantiated, we are able to study the behaviour of the system at a particular instant.

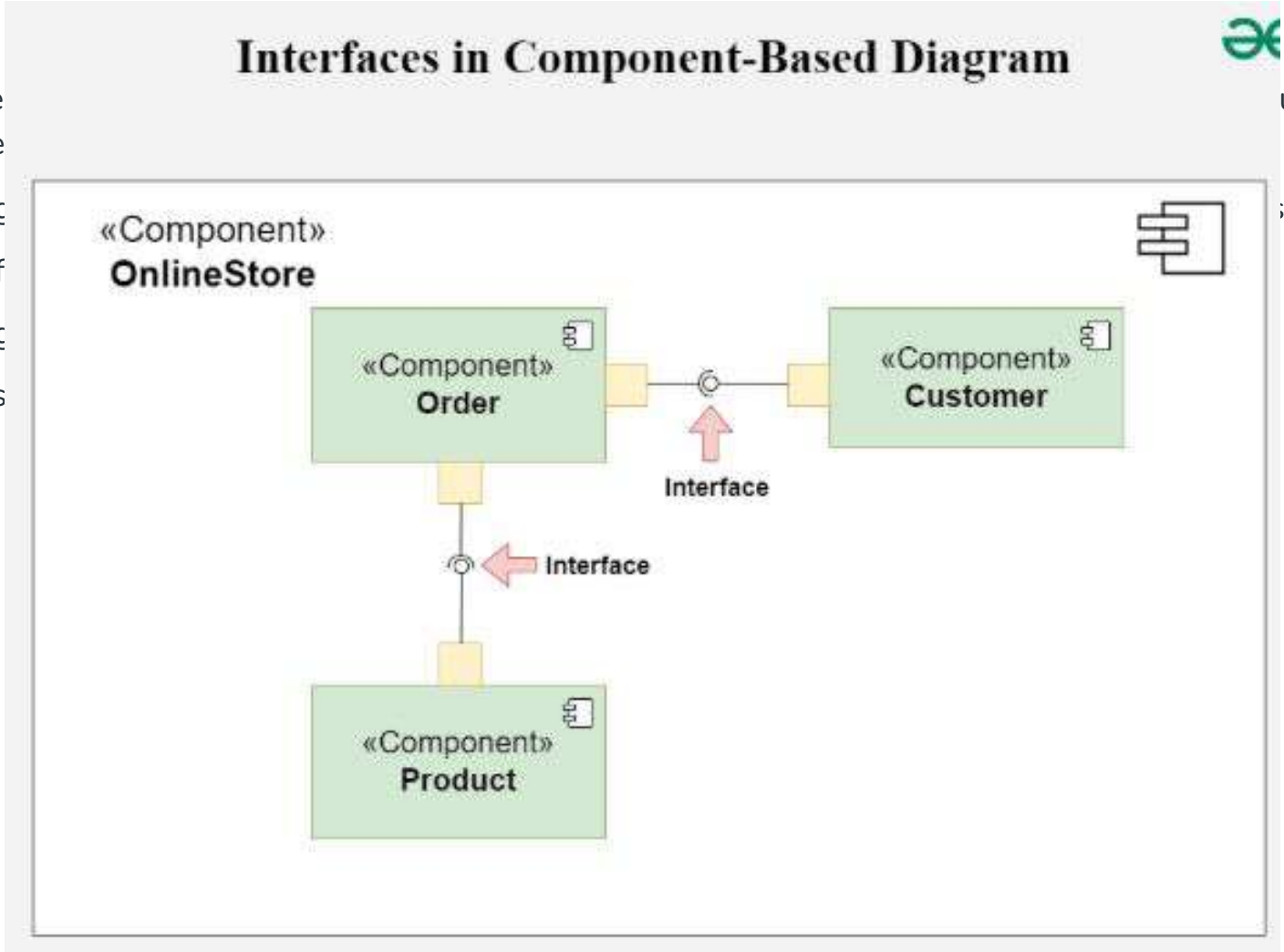
- An object diagram is similar to a class diagram except it shows the instances of classes in the system.
- We depict actual classifiers and their relationships making the use of class diagrams.
- On the other hand, an Object Diagram represents specific instances of classes and relationships between

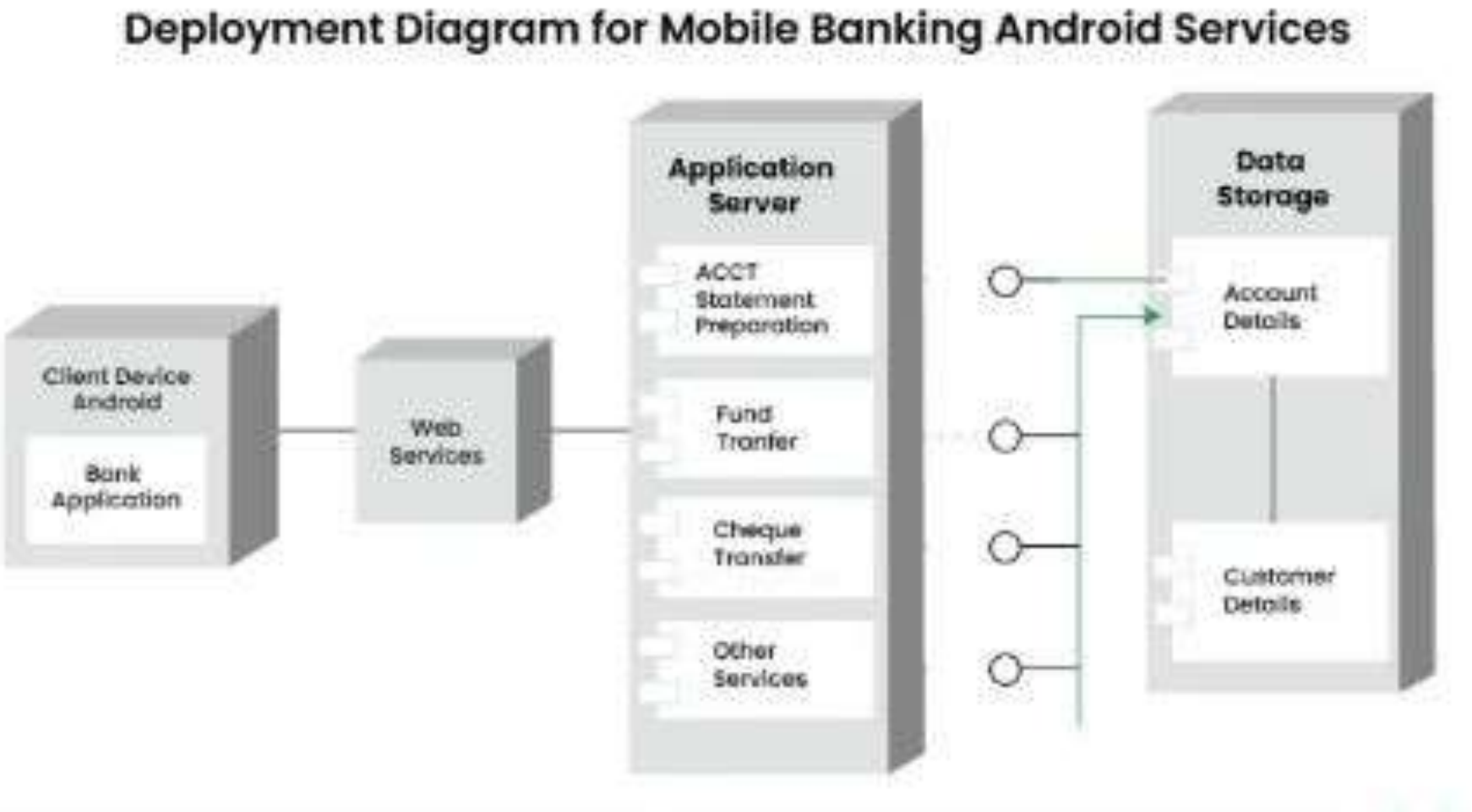


Component Diagram

Component diagrams are
them for modelling imple

- Component Diag
  understanding if
- Component Diag
- Interfaces are us

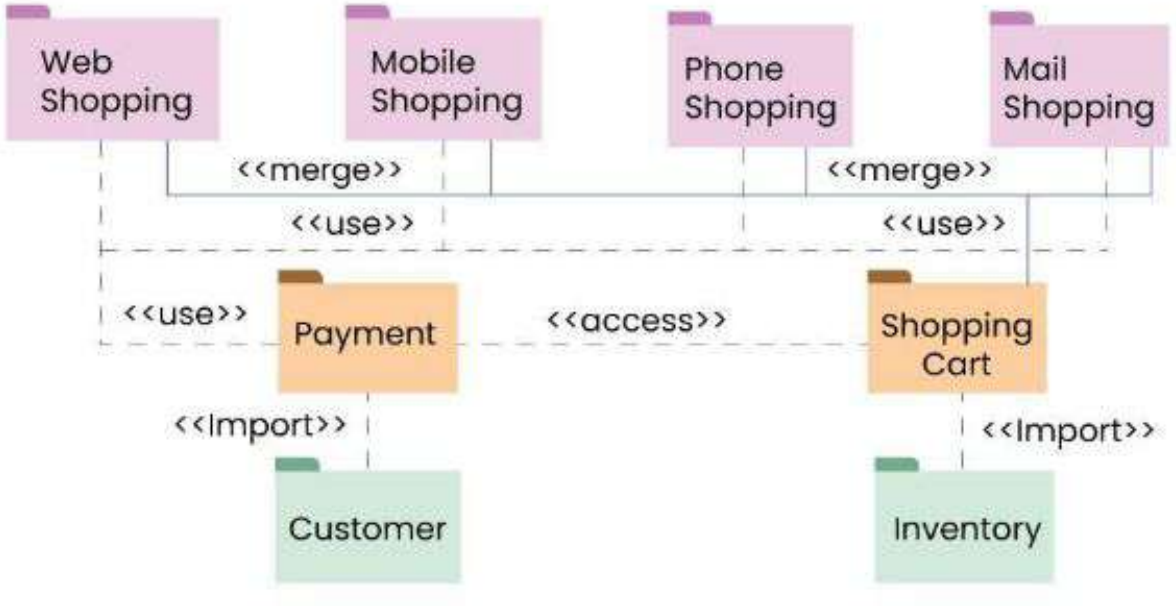



Deployment Diagram for Mobile Banking Android Services
Client Device Android
Bank Application
Web Services
Application Server
ACCT Statement Preparation
Fund Tranfer
Cheque Transfer
Other Services
Data Storage
Account Details
Customer Details

Package Diagram

We use Package Diagrams to depict how packages and their elements have been organized. A package diagram simply shows us the dependencies between different packages and internal composition of packages.

- Packages help us to organise UML diagrams into meaningful groups and make the diagram easy to understand.
- They are primarily used to organise class and use case diagrams.



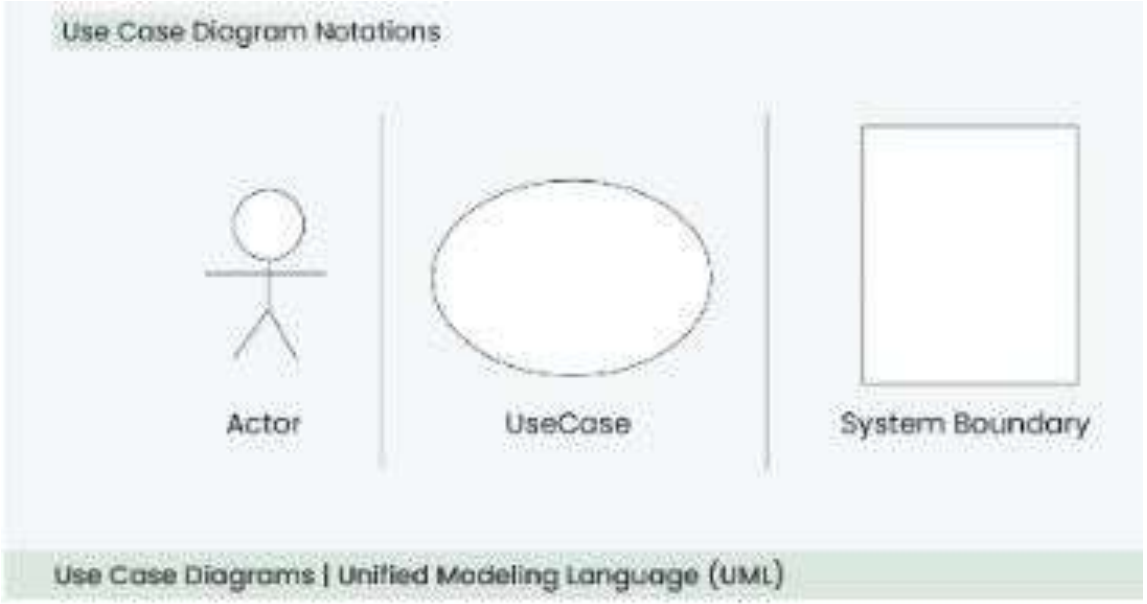
Use Case Diagram - Unified Modeling Language (UML)

- A Use Case Diagram in Unified Modeling Language (UML) is a visual representation that illustrates the interactions between users (actors) and a system.
- It captures the functional requirements of a system, showing how different users engage with various use cases, or specific functionalities, within the system.
- Use case diagrams provide a high-level overview of a system's behavior, making them useful for stakeholders, developers, and analysts to understand how a system is intended to operate from the user's perspective, and how different processes relate to one another.
- They are crucial for defining system scope and requirements.

What is a Use Case Diagram in UML?

A Use Case Diagram is a type of Unified Modeling Language (UML) diagram that represents the interaction between actors (users or external systems) and a system under consideration to accomplish specific goals.

It provides a high-level view of the system's functionality by illustrating the various ways users can interact with it.



When to apply Use Case Diagram?

Use case diagrams are useful in several situations. Here's when you should consider using them:

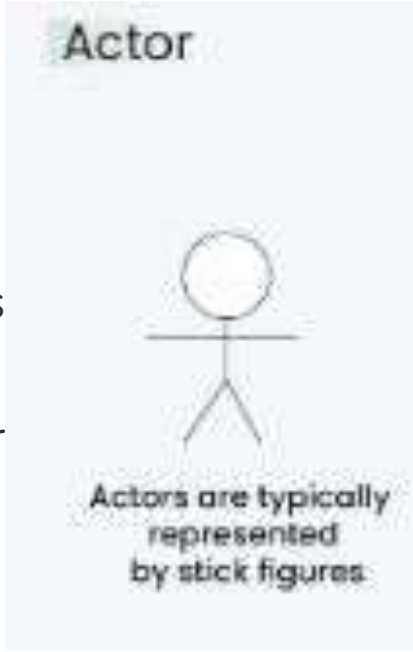
- When you need to gather and clarify user requirements, use case diagrams help visualize how different users interact with the system.
- If you're working with diverse groups, including non-technical stakeholders, these diagrams provide a clear and simple way to convey system functionality.
- During the system design phase, use case diagrams help outline user interactions and plan features, ensuring that the design aligns with user needs.
- When defining what is included in the system versus what is external, use case diagrams help clarify these boundaries.

Use Case Diagram Notations

- UML notations provide a visual language that enables software developers, designers, and other stakeholders to communicate and document system designs, architectures, and behaviors in a consistent and understandable manner.

1. Actors

- Actors are external entities that interact with the system.
- These can include users, other systems, or hardware devices.
- In the context of a Use Case Diagram, actors initiate use cases and receive the outcomes.
- Proper identification and understanding of actors are crucial for accurately modeling system behavior.



2. Use Cases

- Use cases are like scenes in the play.
- They represent specific things your system can do.
- In the online shopping system, examples of use cases could be
  - "Place Order,"
  - "Track Delivery," or
  - "Update Product Information".
- Use cases are represented by ovals.

3. System Boundary

- The system boundary is a visual representation of the scope or limits of the system you are modeling.
- It defines what is inside the system and what is outside.
- The boundary helps to establish a clear distinction between the elements that are part of the system and those that are external to it.
- The system boundary is typically represented by a rectangular box that surrounds all the use cases of the system.
- *The purpose of system boundary is to clearly outlines the boundaries of the system, indicating which components are internal to the system and which are external actors or entities interacting with the system.*

Use Case Diagram Relationships

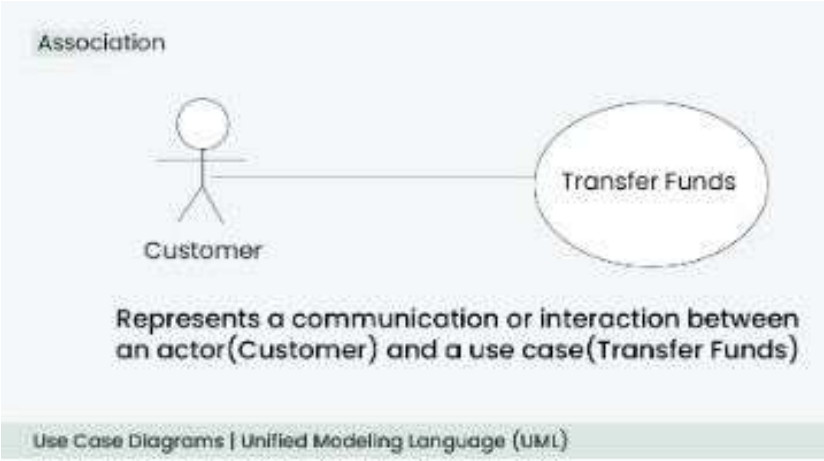
- In a Use Case Diagram, relationships play a crucial role in depicting the interactions between actors and use cases.
- These relationships provide a comprehensive view of the system's functionality and its various scenarios.
- Let's delve into the key types of relationships and explore examples to illustrate their usage.

1. Association Relationship

The Association Relationship represents a communication or interaction between an actor and a use case.

It is depicted by a line connecting the actor to the use case.

This relationship signifies that the actor is involved functionality described by the use case.



Example: Online Banking System

- **Actor:** Customer
- **Use Case:** Transfer Funds
- **Association:** A line connecting the "Customer" actor to the "Transfer Funds" use case, indicating the customer's involvement in the funds transfer process.

2. Include Relationship

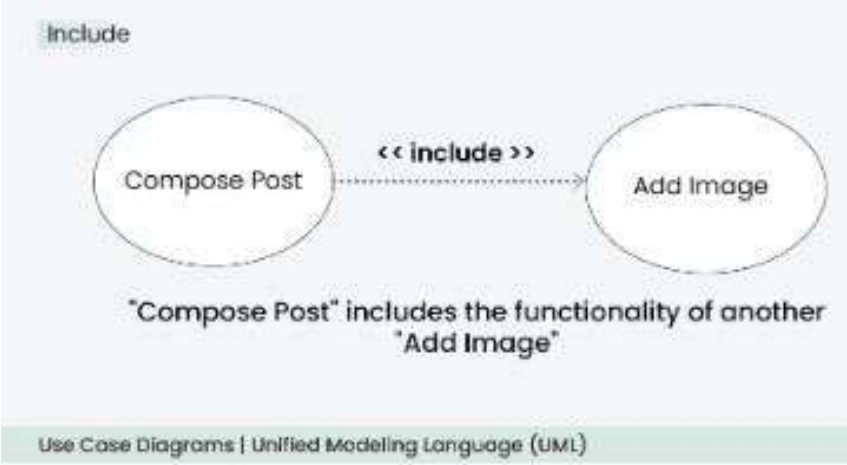
The Include Relationship indicates that a use case includes the functionality of another use case.

It is denoted by a dashed arrow pointing from the including use case to the included use case.

This relationship promotes modular and reusable design.

Example: Social Media Posting

- **Use Cases:** Compose Post, Add Image
- **Include Relationship:** The "Compose Post" use case includes the functionality of "Add Image." Therefore, composing a post includes the action of adding an image.

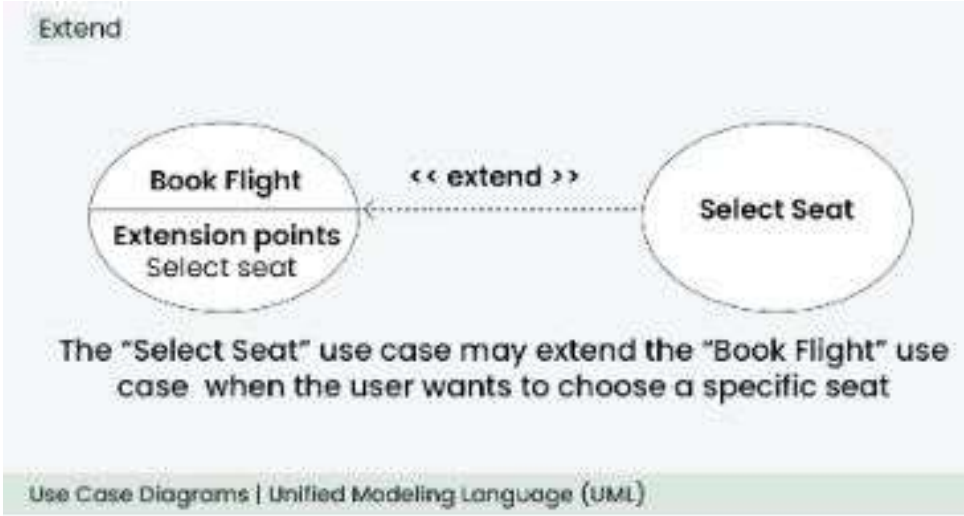


3. Extend Relationship

The Extend Relationship illustrates that a use case can be extended by another use case under specific conditions.

It is represented by a dashed arrow with the keyword "extend."

This relationship is useful for handling optional or exceptional behavior.



Example: Flight Booking System

- **Use Cases:** Book Flight, Select Seat
- **Extend Relationship:** The "Select Seat" use case may extend the "Book Flight" use case when the user wants to choose a specific seat, but it is an optional step.

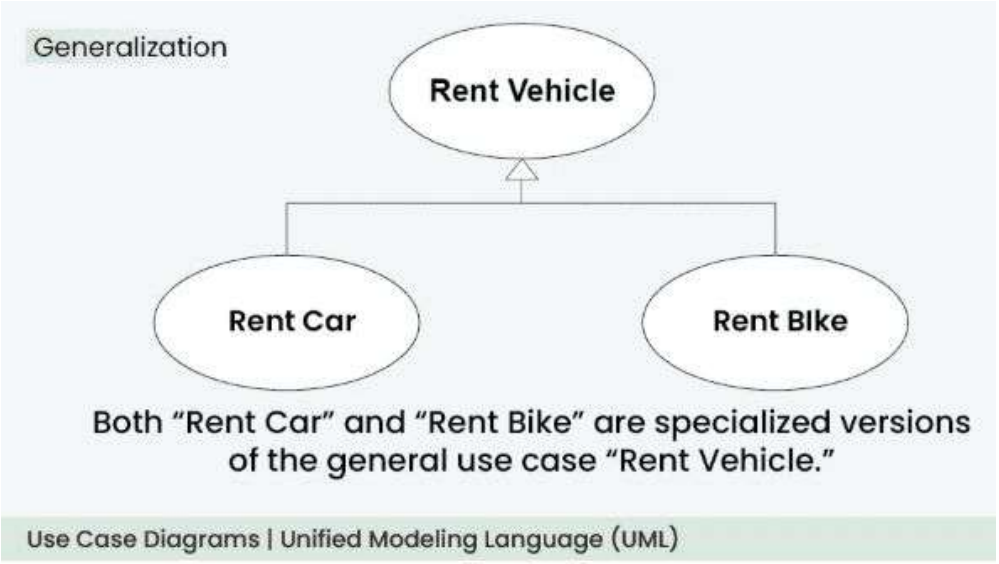
4. Generalization Relationship

The Generalization Relationship establishes an "is-a" connection between two use cases, indicating that one use case is a specialized version of another.

It is represented by an arrow pointing from the specialized use case to the general use case.

Example: Vehicle Rental System

- **Use Cases:** Rent Car, Rent Bike
- **Generalization Relationship:** Both "Rent Car" and "Rent Bike" are specialized versions of the general use case "Rent Vehicle."



How to draw a Use Case diagram in UML?

Below are the main steps to draw use case diagram in UML:

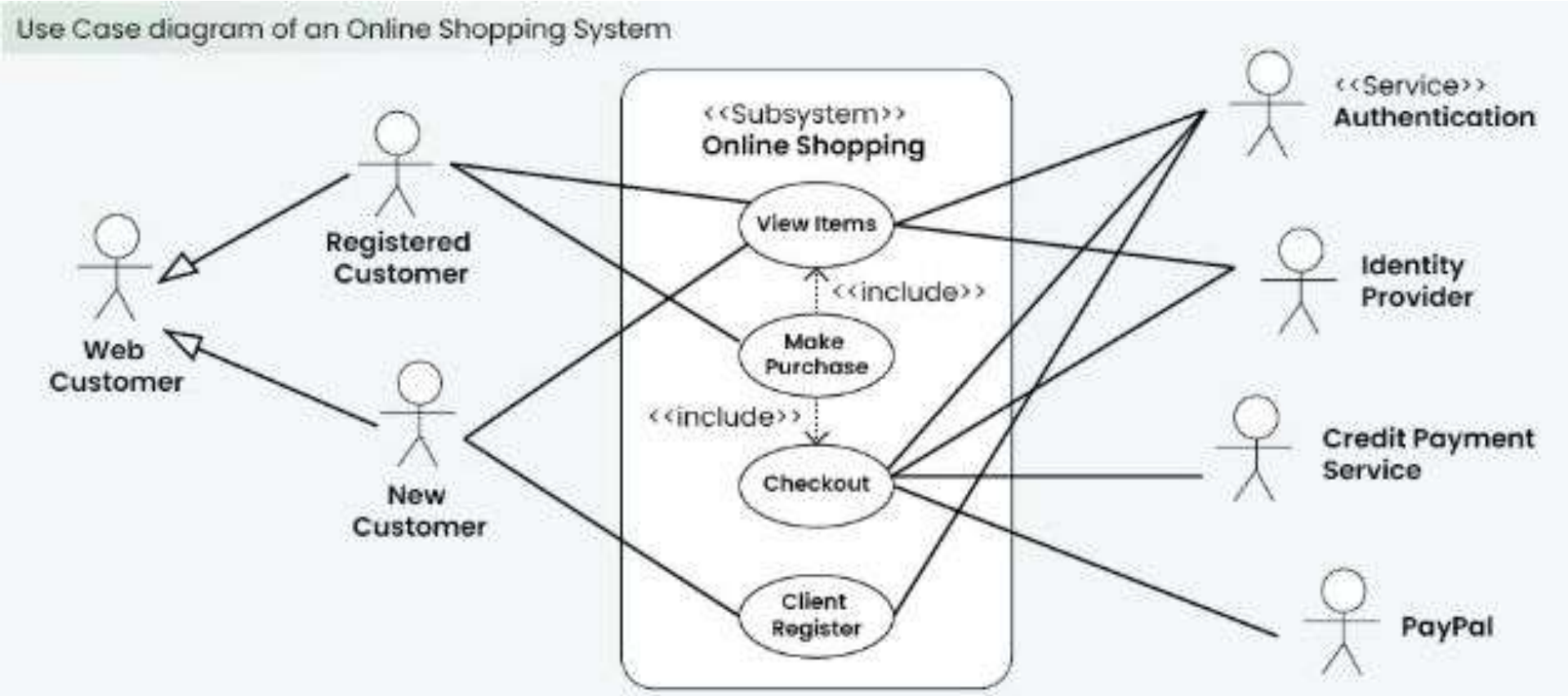
- **Step 1: Identify Actors**: Determine who or what interacts with the system. These are your actors. They can be users, other systems, or external entities.
- **Step 2: Identify Use Cases**: Identify the main functionalities or actions the system must perform. These are your use cases. Each use case should represent a specific piece of functionality.
- **Step 3: Connect Actors and Use Cases**: Draw lines (associations) between actors and the use cases they are involved in. This represents the interactions between actors and the system.
- **Step 4: Add System Boundary**: Draw a box around the actors and use cases to represent the system boundary. This defines the scope of your system.
- **Step 5: Define Relationships**: If certain use cases are related or if one use case is an extension of another, you can indicate these relationships with appropriate notations.
- **Step 6: Review and Refine**: Step back and review your diagram. Ensure that it accurately represents the interactions and relationships in your system. Refine as needed.
- **Step 7: Validate**: Share your use case diagram with stakeholders and gather feedback. Ensure that it aligns with their understanding of the system's functionality.

Use Case Diagram example(Online Shopping System)

Let's understand how to draw a Use Case diagram with the help of an Online Shopping System:

- **Actors:**
    - Customer
    - Admin
- **Use Cases:**
    - Browse Products
    - Add to Cart
    - Checkout
    - Manage Inventory (Admin)
- **Relations:**
    - The Customer can browse products, add to the cart, and complete the checkout.
    - The Admin can manage the inventory.

Below is the use case diagram of an Online Shopping System:



What is the Purpose and Benefits of Use Case Diagrams?

The Use Case Diagram offers numerous benefits throughout the system development process. Here are some key advantages of using Use Case Diagrams:

- Use Case Diagrams offer a clear visual representation of a system's functions and its interactions with external users.

  This representation helps stakeholders, including those without technical expertise, in grasping the system's overall behavior.

- They establish a shared language for articulating system requirements, ensuring that all team members have a common understanding.
- Use Case Diagram illustrate the different ways users engage with the system, contributing to a thorough comprehension of its functionalities.
- In the design phase, Use Case Diagrams help outline how users (actors) will interact with the system.

  They support the planning of user interfaces and aid in structuring system functionalities.

More Example to solve

https://www.uml-diagrams.org/use-case-diagrams-examples.html

Airport Check-In and Security Screening

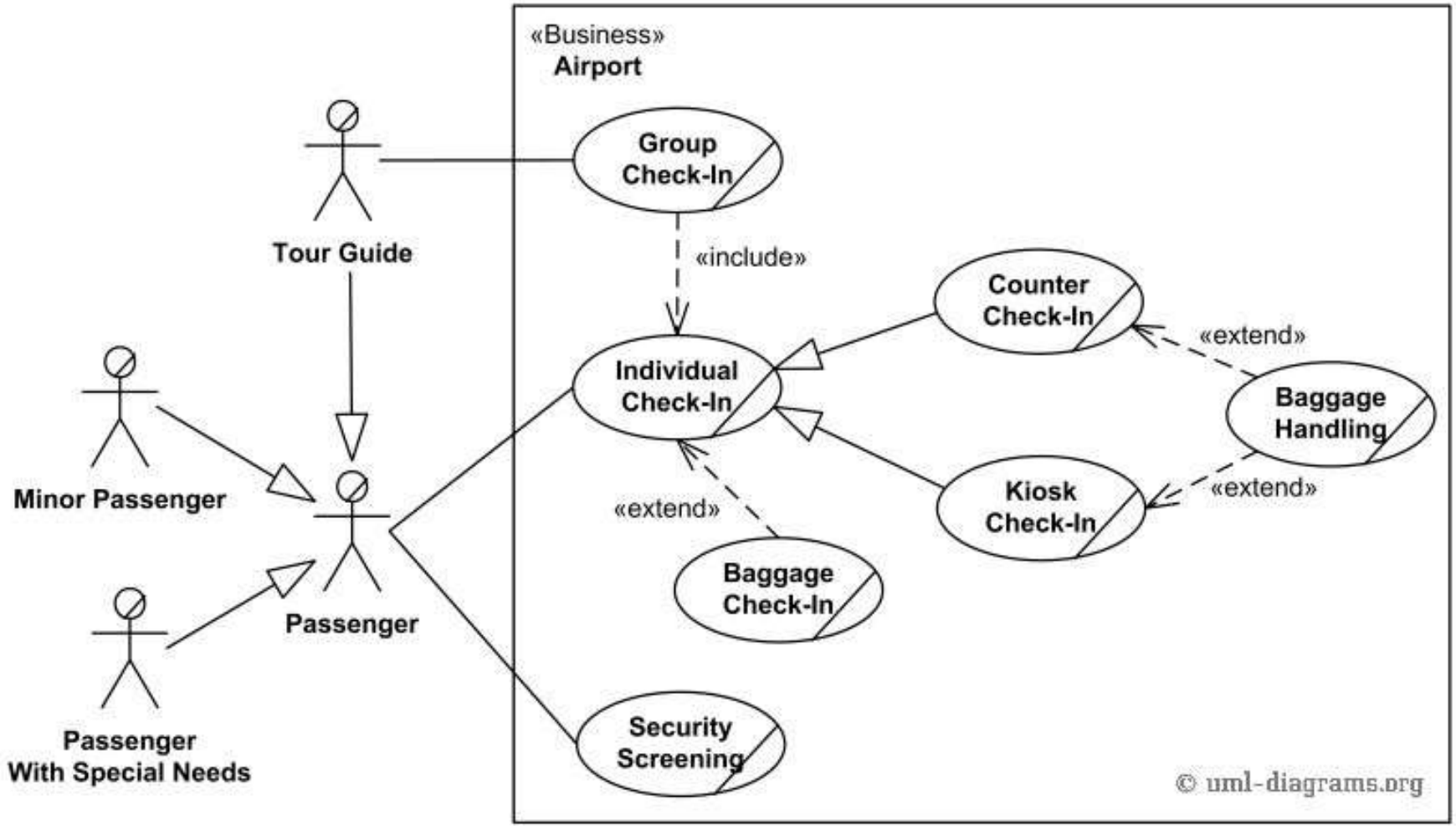
UML Use Case Diagram Example

This is an example of a **business use case** diagram which is usually created during **Business Modeling** and is rendered here in **Rational Unified Process** (RUP) notation.

Business actors are Passenger, Tour Guide, Minor (Child), Passenger with Special Needs (e.g. with disabilities), all playing **external roles** in relation to airport business.

Business use cases are Individual Check-In, Group Check-In (for groups of tourists), Security Screening, etc. - representing business functions or processes taking place in airport and serving the needs of passengers.

Business use cases Baggage Check-in and Baggage Handling extend Check-In use cases, because passenger might have no luggage, so baggage check-in and handling are optional.

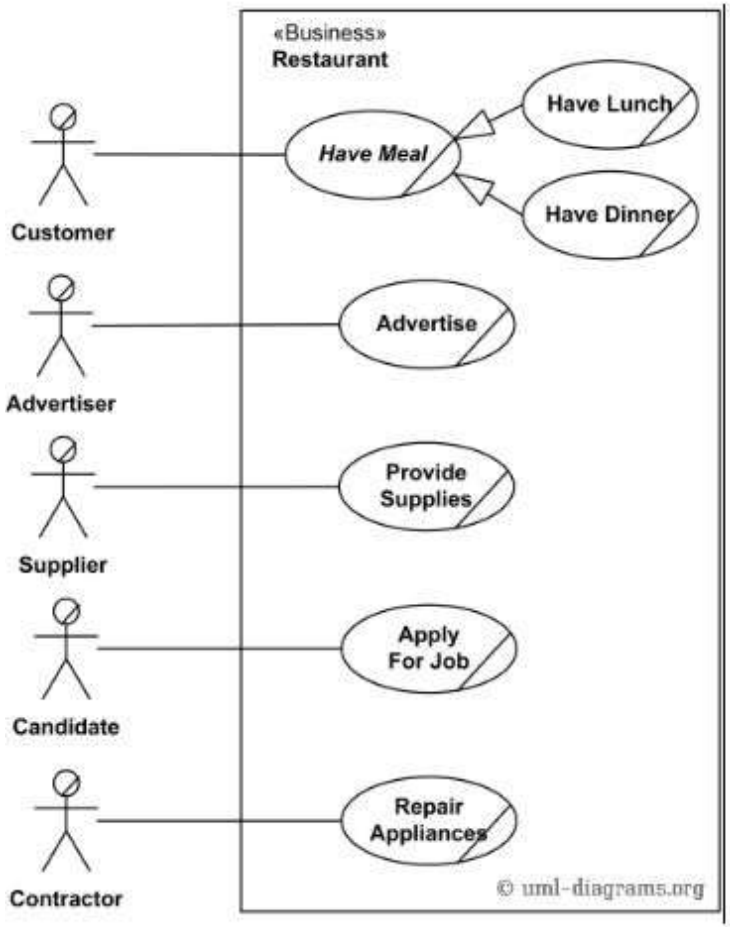

«Business»
Airport
Group Check-In
Tour Guide
«include»
Counter Check-In
«extend»
Individual Check-In
Baggage Handling
Minor Passenger
Kiosk Check-In
«extend»
«extend»
Baggage Check-In
Passenger
Passenger With Special Needs
Security Screening
© uml-diagrams.org

Restaurant

UML Use Case Diagram Example

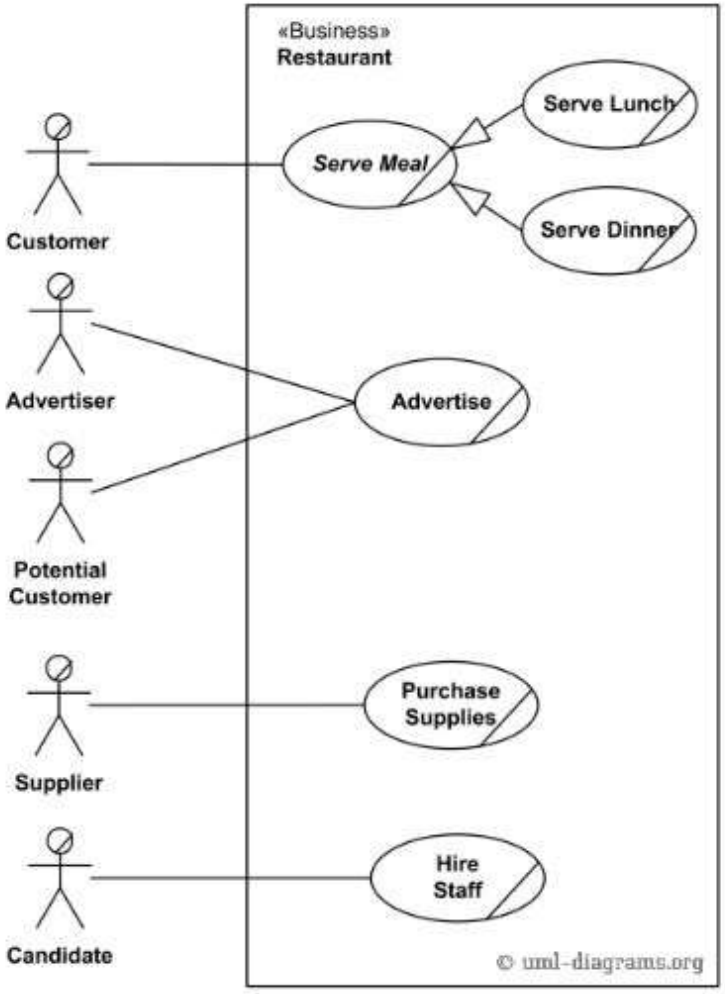
Here we provide two alternative examples of a **business use case** diagram for a Restaurant, rendered in a notation used by **Rational Unified Process** (RUP).

First example shows **external business view** of a restaurant. We can see several **business actors** having some needs and goals as related to the restaurant and **business use cases** expressing expectations of the actors from the business.



Business use case diagram for Restaurant - External view

- For example, **Customer** wants to **Have Meal**, **Candidate** - to **Apply for Job**, and **Contractor** - to fix some appliances. Note, that we don't have such actors as **Chef** or **Waiter**.
- They are not external roles but part of the business we model - the Restaurant, thus - they are not actors. In terms of RUP Chef and Waiter are **business workers**.
- Second example shows **internal business view** of a restaurant. In this case we can see that restaurant has several business processes represented by **business use cases** which provide some services to external **business actors**.
- As in the previous example, actors have some needs and goals as related to the restaurant.
- This approach could be more useful to model services that the business provides to different types of customers, but reading this kind of business use case diagrams could be confusing.
- For example, **Customer** is now connected to **Serve Meal** use case, **Supplier** - to **Purchase Supplies**.
- We have now new actor **Potential Customer** participating in **Advertise** use case by reading ads and getting some information about restaurant.
- At the same time, **Contractor** actor is gone because **Repair Appliances** is not a service usually provided by restaurants.



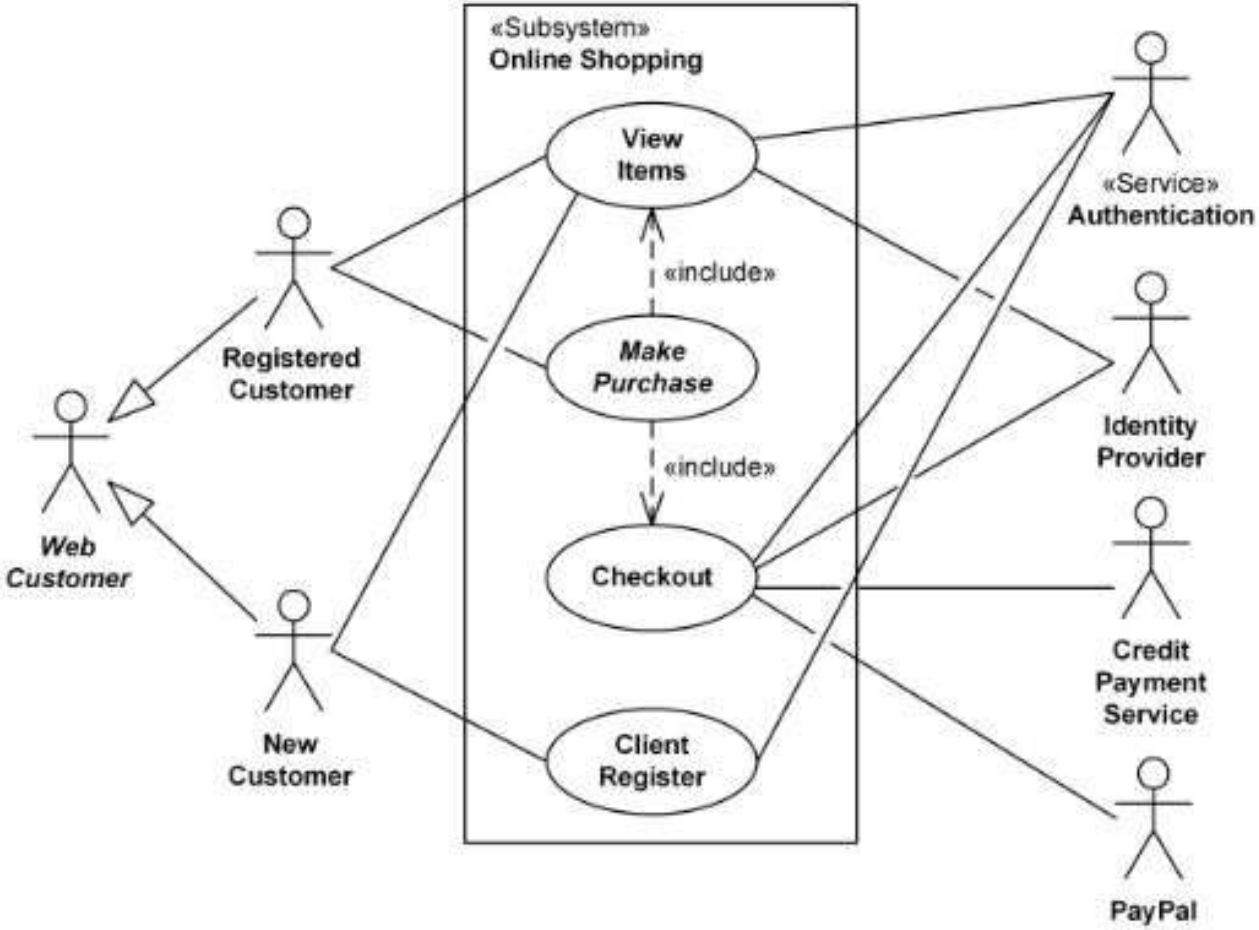
Still, in this example we don't have actors as **Chef** or **Waiter** for the same reasons as before - they both are not external roles but part of the business we model.

Online Shopping

UML Use Case Diagram Example

Web Customer **actor** uses some web site to make purchases online. Top level **use cases** are **View Items**, **Make Purchase** and **Client Register**. View Items use case could be used by customer as top level use case if customer only wants to find and see some products. This use case could also be used as a part of Make Purchase use case. Client Register use case allows customer to register on the web site, for example to get some coupons or be invited to private sales. Note, that **Checkout** use case is **included use case** not available by itself - checkout is part of making purchase.

Except for the **Web Customer** actor there are several other actors which will be described below with detailed use cases.



Online shopping UML use case diagram example - top level use cases.

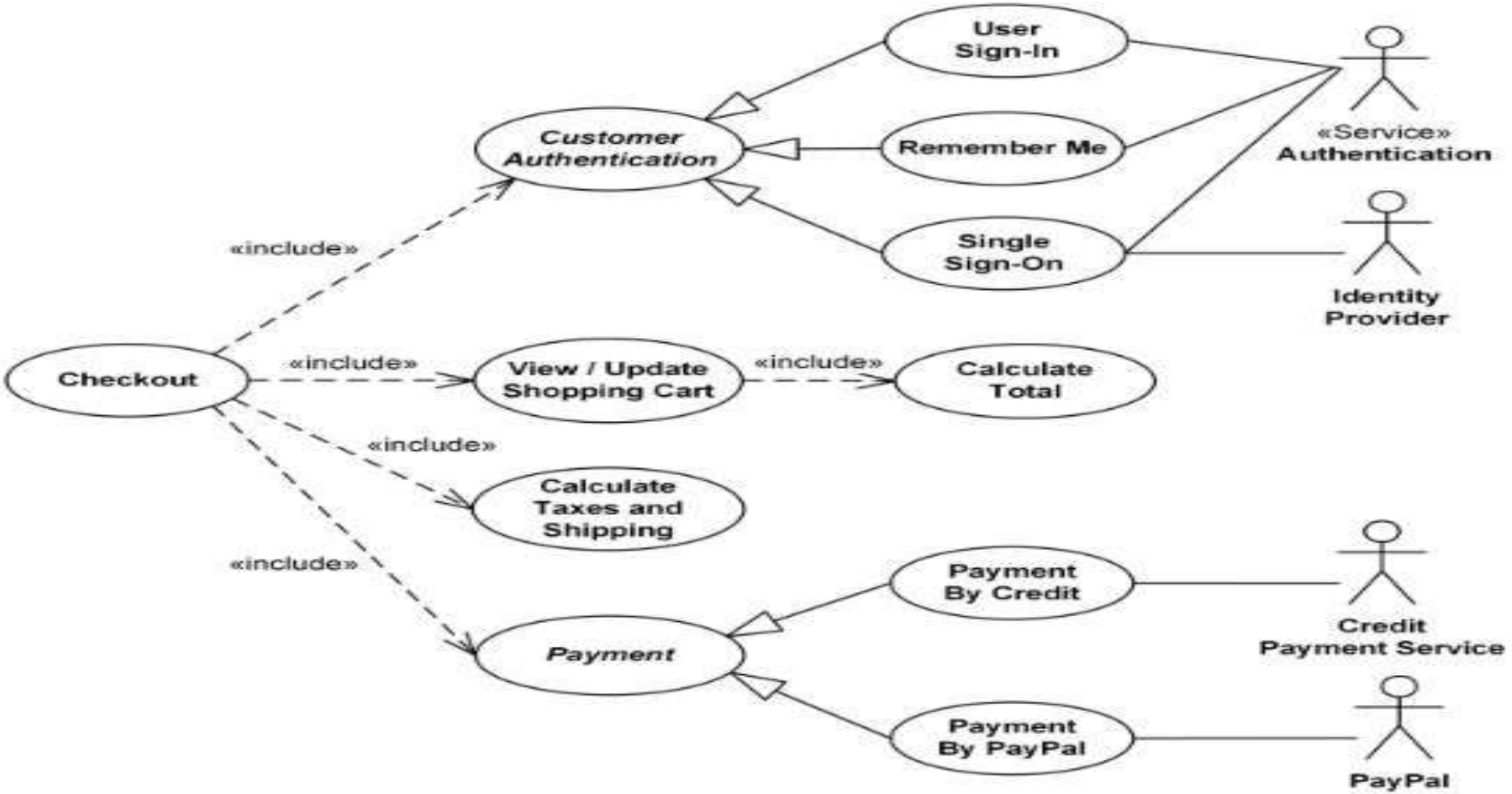
View Items use case is **extended** by several optional use cases - customer may search for items, browse catalog, view items recommended for him/her, add items to shopping cart or wish list. All these use cases are extending use cases because they provide some optional functions allowing customer to find item.

Customer Authentication use case is **included** in **View Recommended Items** and **Add to Wish List** because both require the customer to be authenticated. At the same time, item could be added to the shopping cart without user authentication.

Online shopping UML use case diagram example - view items use case.

Checkout use case includes several required uses cases. Web customer should be authenticated. It could be done through user login page, user authentication cookie ("Remember me") or Single Sign-On (SSO). Web site authentication service is used in all these use cases, while SSO also requires participation of external identity provider.

Checkout use case also includes **Payment** use case which could be done either by using credit card and external credit payment service or with PayPal.



Online shopping UML use case diagram example - checkout, authentication and payment use cases.

Behavioral UML Diagrams

Behavioral UML diagrams are visual representations that depict the dynamic aspects of a system, illustrating how objects interact and behave over time in response to events.

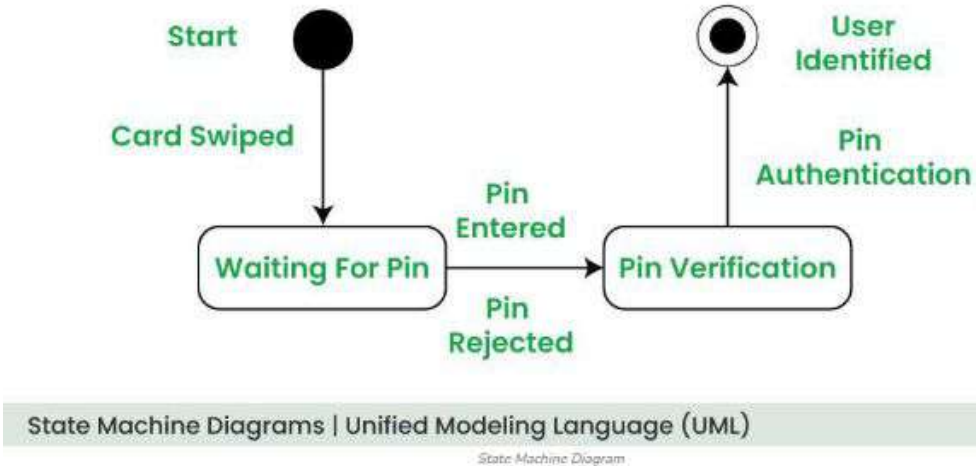
State Machine Diagrams

A state diagram is used to represent the condition of the system or part of the system at finite instances of time.

It's a behavioral diagram and it represents the behavior using finite state transitions.

- State diagrams are also referred to as **State machines** and **State-chart Diagrams**
- These terms are often used interchangeably.
- So simply, a state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.

A State Machine Diagram for user verification

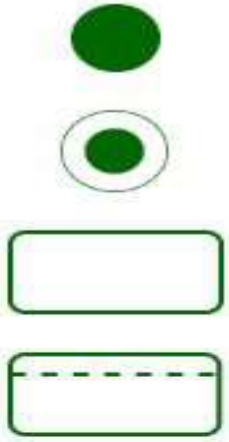


State Transition Diagram

- **State Transition Diagram** are also known as Dynamic models.
- As the name suggests, it is a type of diagram that is used to represent different transition (changing) states of a System.
- It is generally used to graphically represent all possible transition states a system can have and model such systems.
- It is very essential and important and right for object-oriented modeling from the beginning.
- The System consists of various states that are being represented using various symbols in the state transition diagram.

- **You can see the symbols and their description given below :**

1. Initial State -
2. Final State -
3. Simple State -
4. Composite State -



Type of State	Description
Initial State	In a System, it represents Starting state.
Final State	In a System, it represents Ending state.
Simple State	In a System, it represents a Simple state with no substructure.
Composite State	In a System, it represents a Composite state with two or more parallel or concurrent states out of which only one state will be active at a time and other states will be inactive.

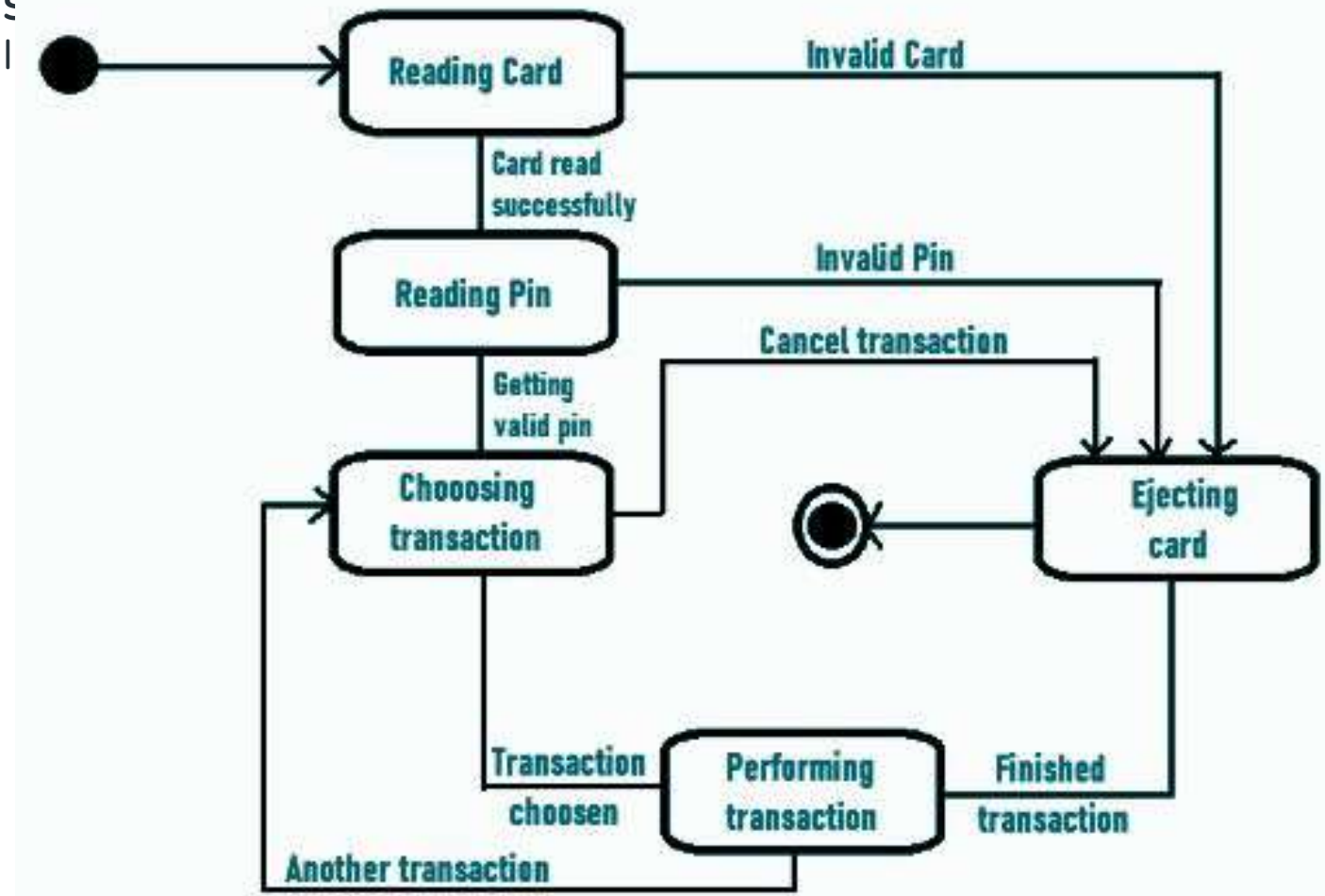
Key Components of a State Transition Diagram

- **States:** Represented by circles or boxes, these are the distinct modes or observable behaviors of a system.
- **Transitions:** Shown as arrows connecting states, these arrows indicate a change from one state to another.
- **Events:** Labels on the arrows, these are the specific occurrences that trigger a transition from one state to the next.
- **Initial State:** Indicated by an arrow pointing to the first state, often a circle with a black dot.
- **Final/End State:** A stop state, typically a circle within a circle.

How it Works

- The system begins in an initial state.
- An event occurs.
- The system transitions from its current state to a new state, as indicated by the arrow associated with that event.
- This process continues, with the system moving through various states in response to a sequence of events.

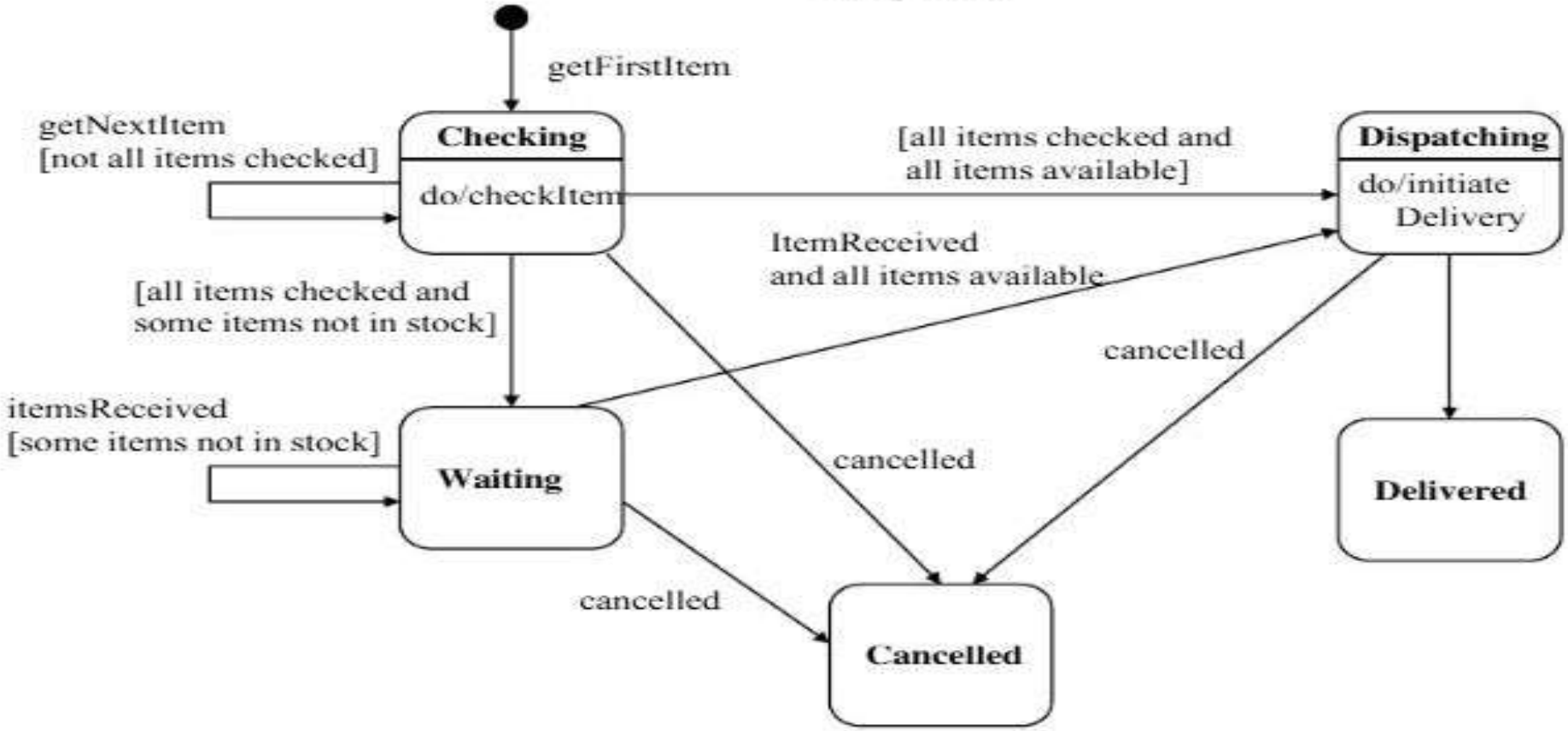
s using ATM card.



State Transition Diagram for ATM System

- When the customer inserts the bank or credit card in the ATM's card reader, the entry action i.e readcard is performed by the ATM machine.
- If the card is not valid then the machine will perform exit action.
- After the card is being read successfully, the ATM machine will ask for Pin.
- Then the customer enters the pin and ATM machine then reads pin.
- If the pin entered is not valid then machine will perform exit action.
- If the pin entered is valid, then the machine further process towards transaction.
- After successful transaction, machine undergoes the exit action i.e., ejectcard that discharges the customer's card.

State Diagram Example: States of an Order object

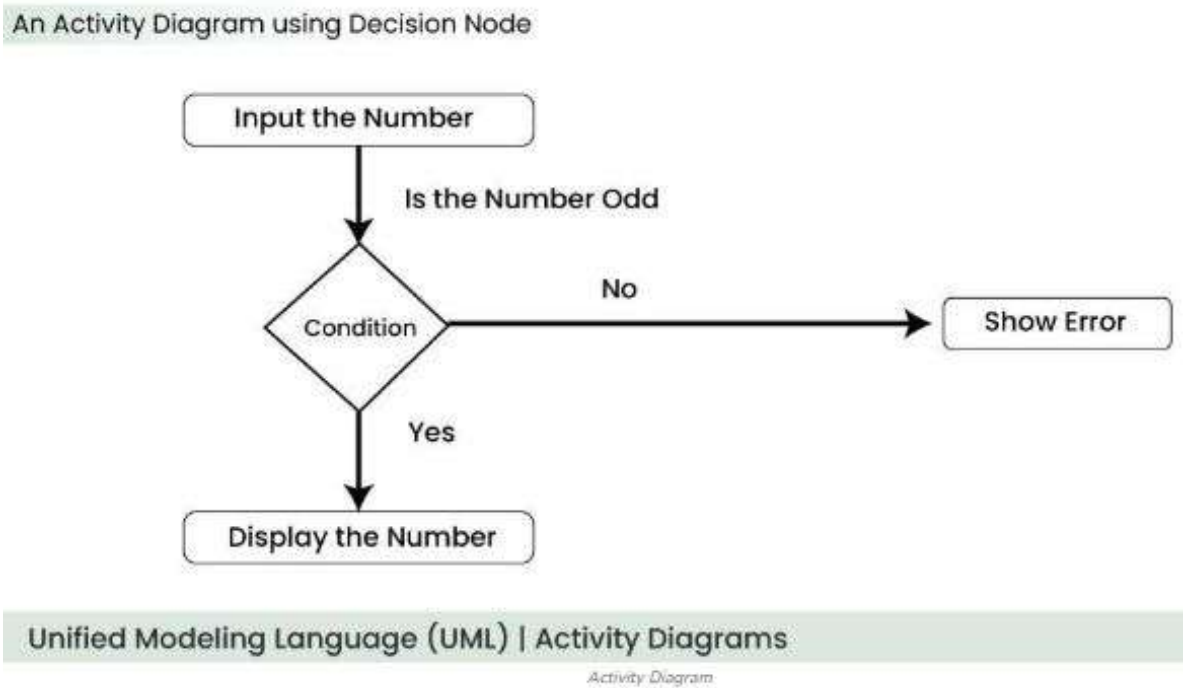


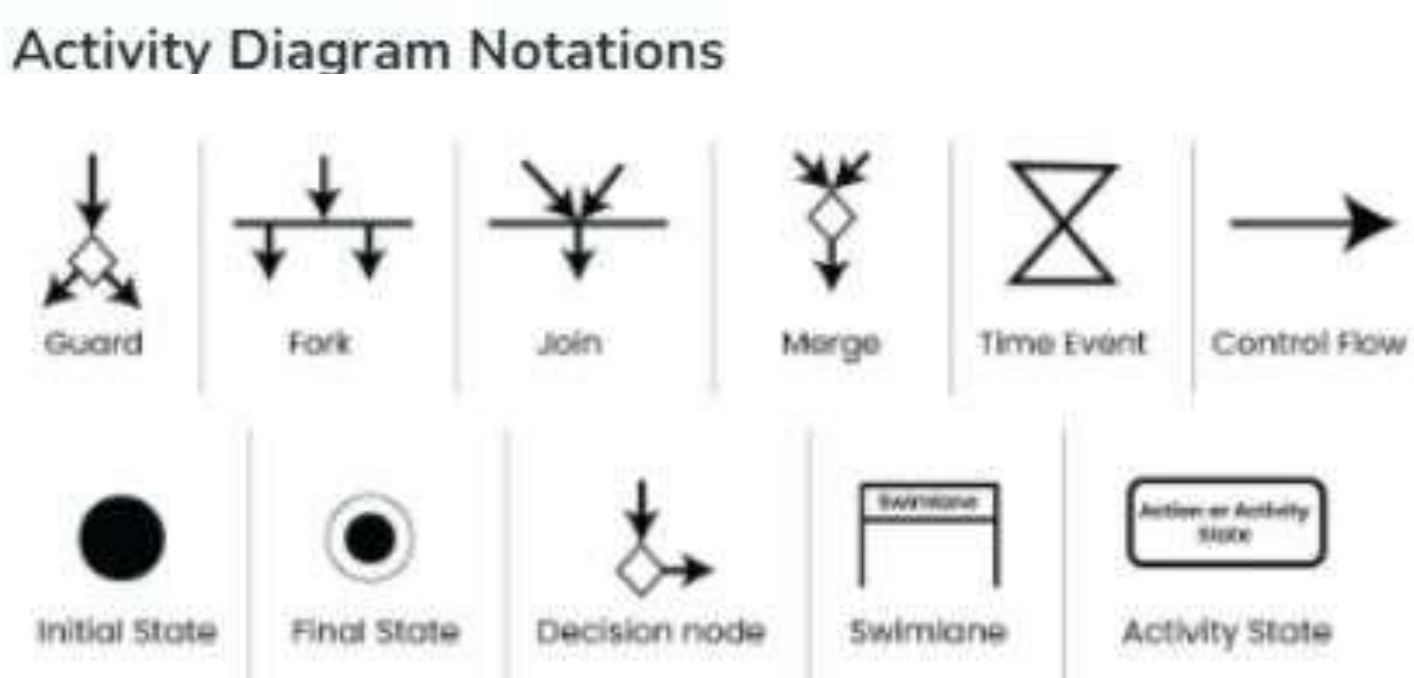
Activity Diagrams

We use Activity Diagrams to illustrate the flow of control in a system.

We can also use an activity diagram to refer to the steps involved in the execution of a use case.

- We model sequential and concurrent activities using activity diagrams.
- So, we basically depict workflows visually using an activity diagram.
- An activity diagram focuses on condition of flow and the sequence in which it happens.
- We describe or depict what causes a particular event using an activity diagram.





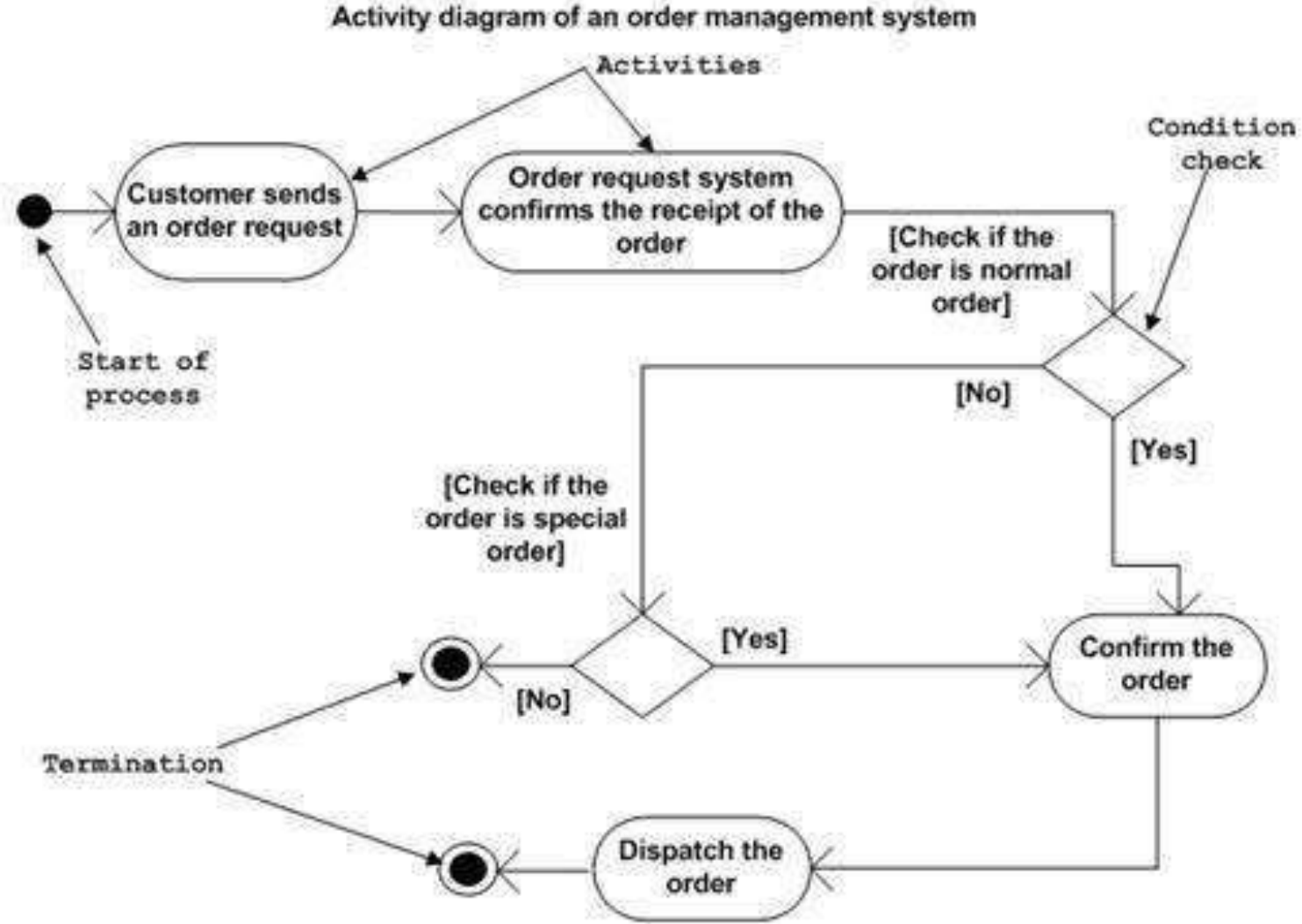
The purpose of an activity diagram can be described as

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

Here are the basic components of an activity diagram:

- **Initial state and final state** – All activity diagrams have an initial state and final state that mark the start and end of the process.
- **Activity or action state** – Represents a single activity that sets a series of actions into motion. An example could be a user logging into their account in a mobile banking system.
- **Actions** – An action or step in the activity in which the system or the user performs a task. Following our example, an action could be a user checking their account balance.
- **Objects** – These are the materials or data that are created or used within an activity.
- **Decisions** – These are the decisions that need to be answered by ‘yes’ or ‘no’ before proceeding to another action or activity.
- **Synchronization** – Consists of the fork node which marks the creation of concurrent flows and the join node which merges back the concurrent flows into a single flow.
- **Signals** – Used to indicate how actions can be made outside the system to modify an activity. For example, in order for payment to push through, a user may receive a signal in the form of an OTP for authorization.
- **Swimlanes** – These are columns or categories that are created to group related activities that are carried out by different actors.

ement system.



Activity diagram can be used for −

- Modeling workflow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigating business requirements at a later stage.

What are the benefits of an activity diagram?

An activity diagram can be helpful in a variety of ways.

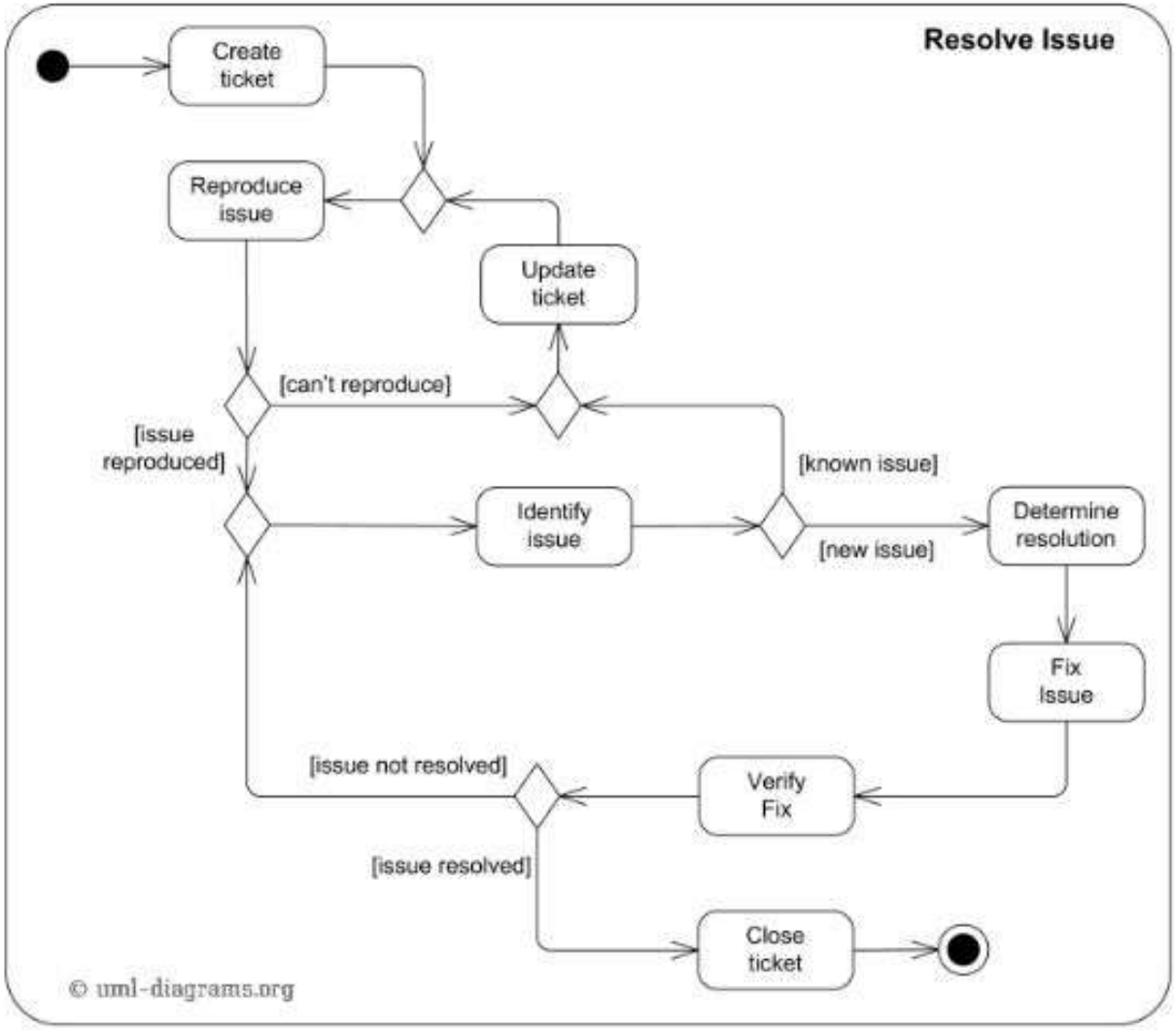
Let’s explore a few ways how activity diagrams benefit users:

- Illustrate the flow of activities so that it’s easy to understand the behavior and structure of a system.
- Allow stakeholders to visualize steps, decisions and interactions to root out inefficiencies.
- Provide a reference point for future developers or those involved in system maintenance.

Resolve Software Issue

UML Activity Diagram Example

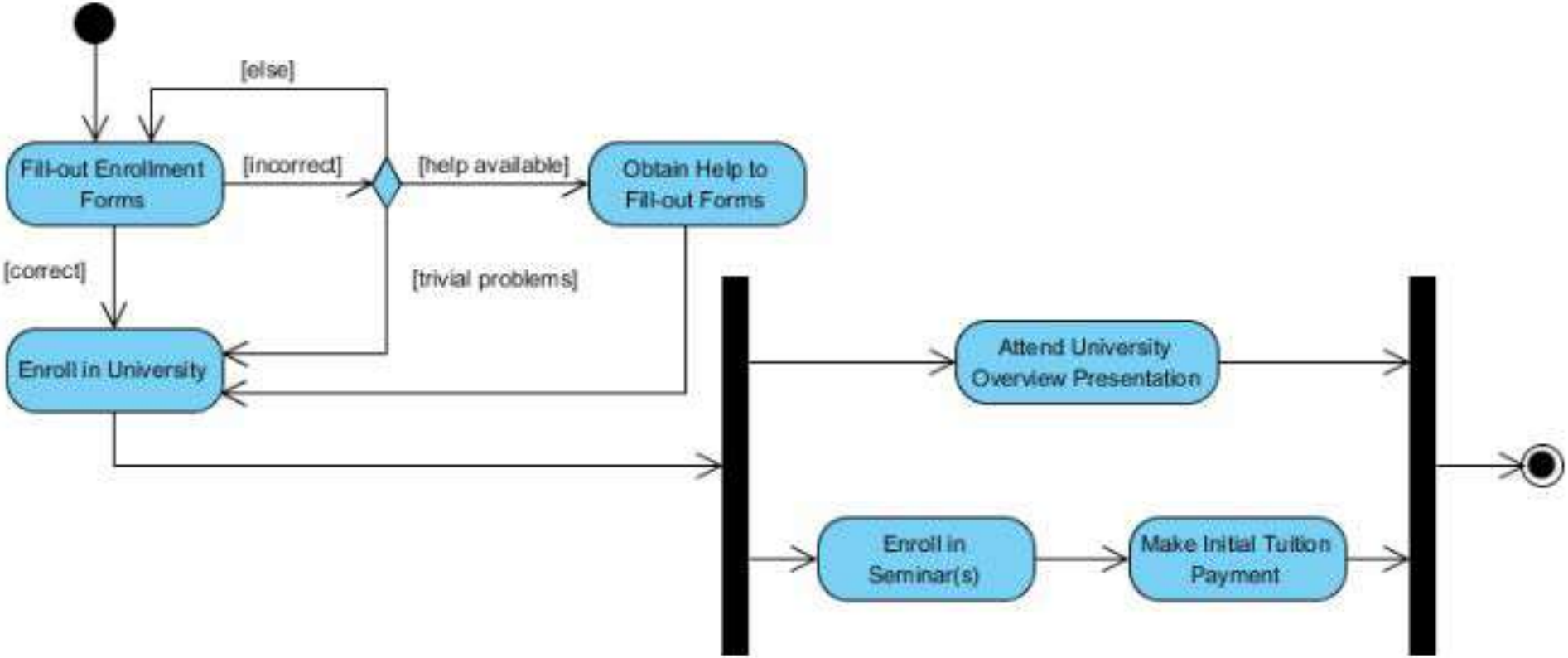
- An example of **UML activity diagram** which shows how to resolve an issue in a software design.
- After ticket is created by some authority and the issue is reproduced, issue is identified, resolution is determined, issue is fixed and verified, and ticket is closed, if issue was resolved.
- This example does not use partitions, so it is not very clear who is responsible for fulfilling each specific action.


Resolve Issue
Create ticket
Reproduce issue
Update ticket
[can't reproduce]
[issue reproduced]
[known issue]
Identify issue
[new issue]
Determine resolution
Fix Issue
[issue not resolved]
Verify Fix
[issue resolved]
Close ticket
© uml-diagrams.org

Activity Diagram Example - Student Enrollment

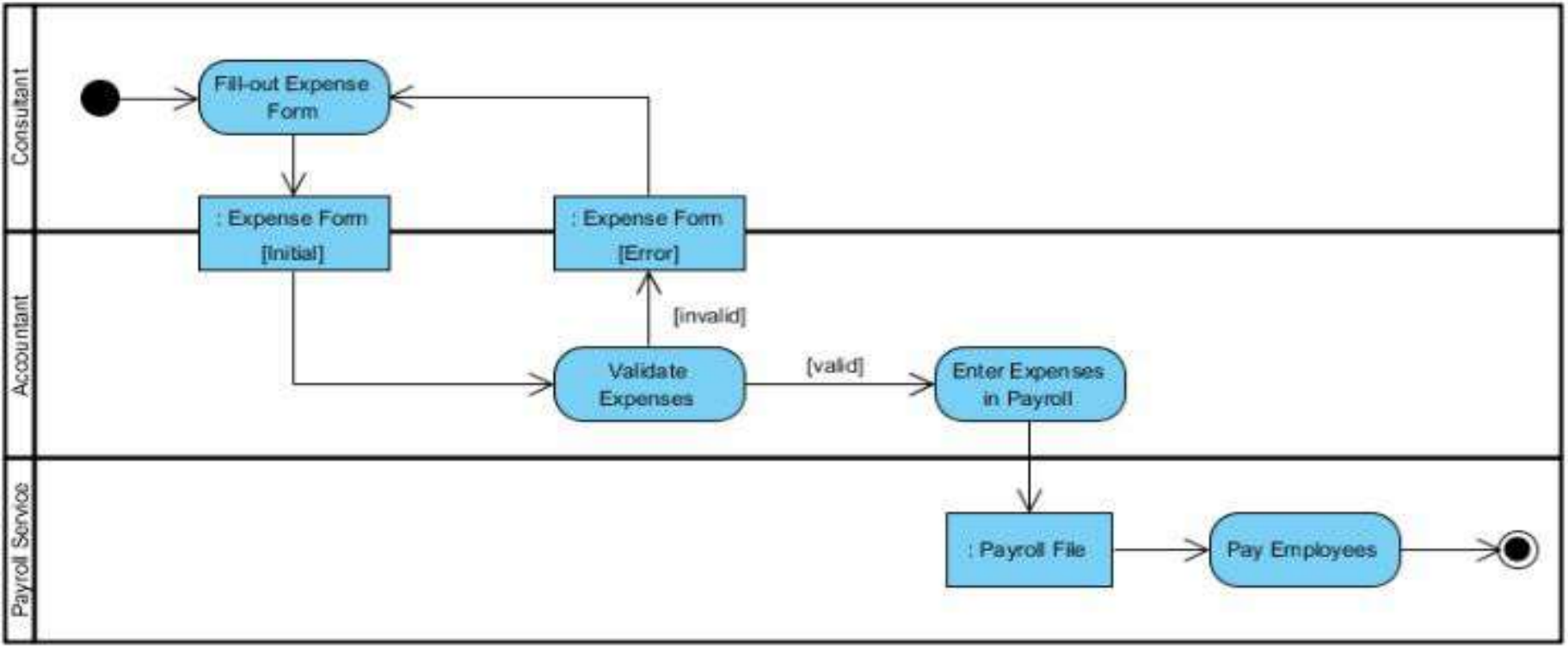
This UML activity diagram example describes a process for student enrollment in a university as follows:

- An applicant wants to enroll in the university.
- The applicant hands a filled out copy of Enrollment Form.
- The registrar inspects the forms.
- The registrar determines that the forms have been filled out properly.
- The registrar informs student to attend in university overview presentation.
- The registrar helps the student to enroll in seminars
- The registrar asks the student to pay for the initial tuition.


[else]
Fill-out Enrollment Forms
[incorrect]
[help available]
Obtain Help to Fill-out Forms
[correct]
[trivial problems]
Enroll in University
Attend University Overview Presentation
Enroll in Seminar(s)
Make Initial Tuition Payment

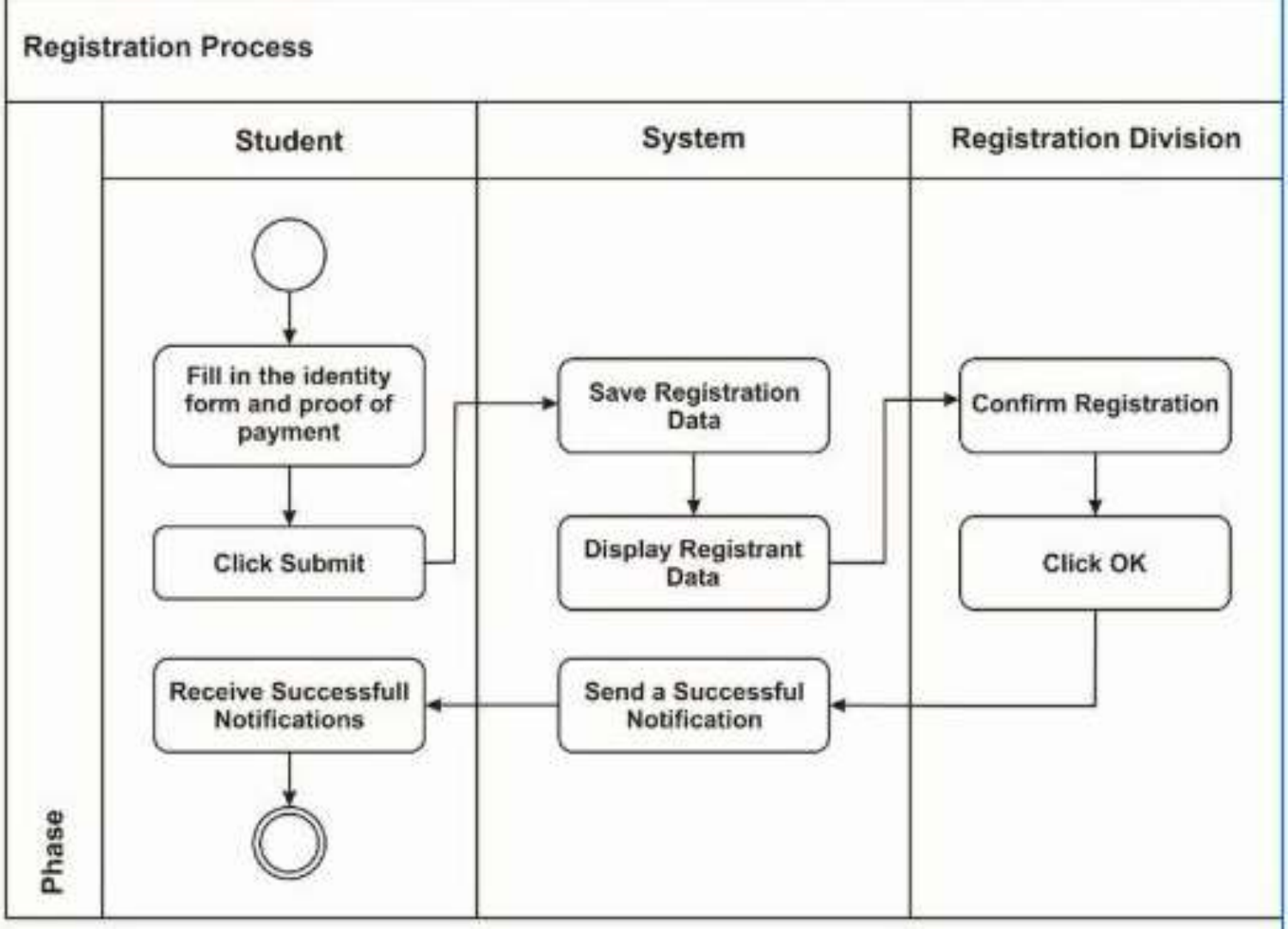
Activity Diagram - Swimlane

A swimlane is a way to group activities performed by the same actor on an activity diagram or activity diagram or to group activities in a single thread. Here is an example of a Swimlane activity diagram for modeling Staff Expenses Submission:



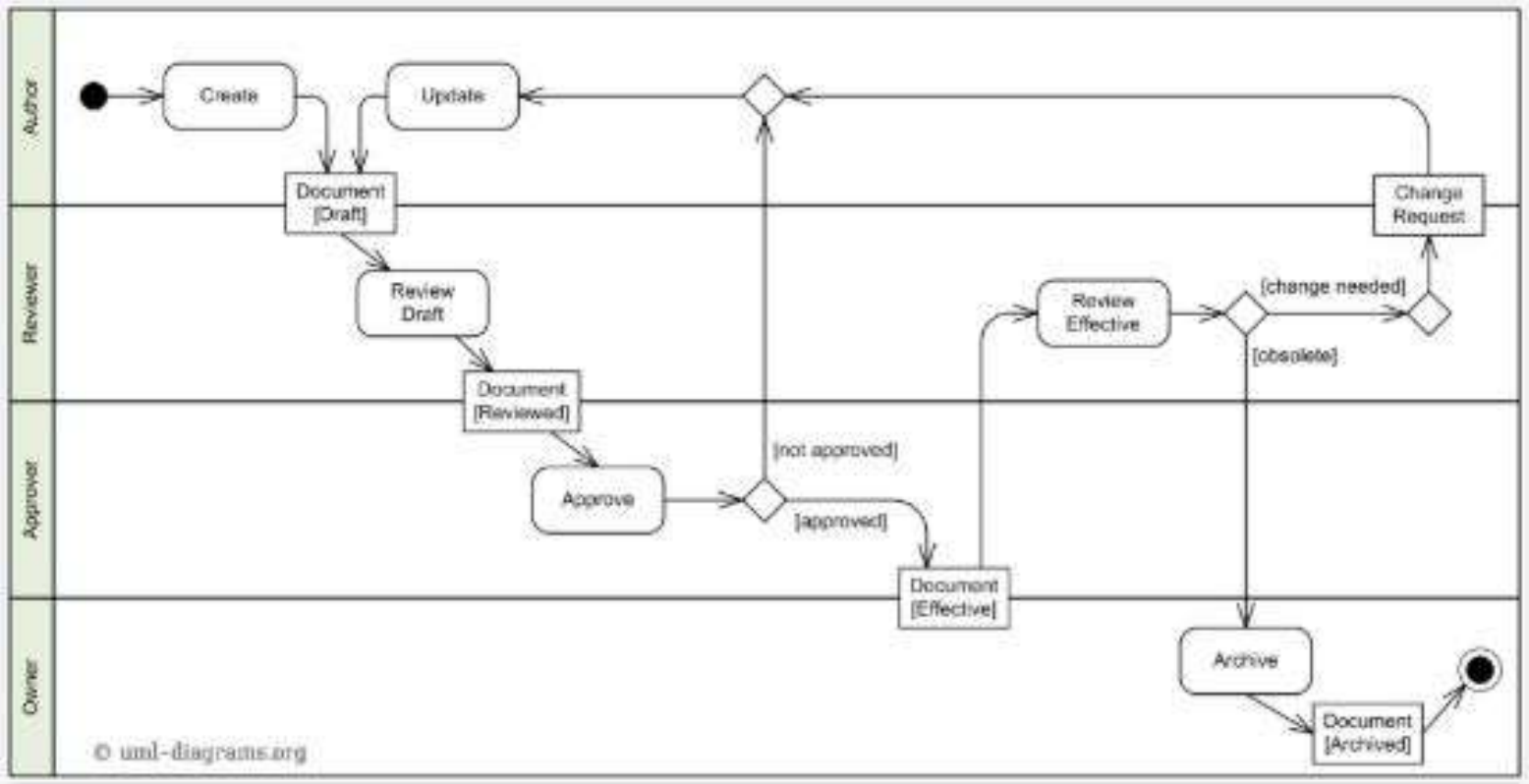
t registration system

https://venngage
gram/



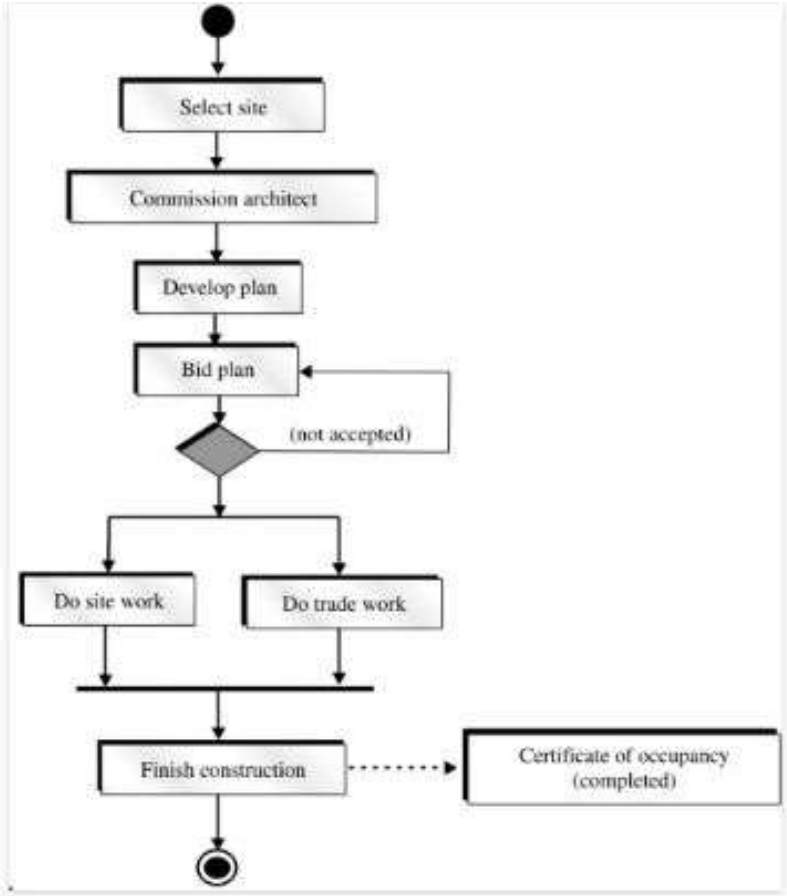
2. Document management process activity diagram

As we have previously mentioned, activity diagrams can also be used to model business processes. In this example, an activity diagram is used to portray the steps involved in the process of creating a formal document:



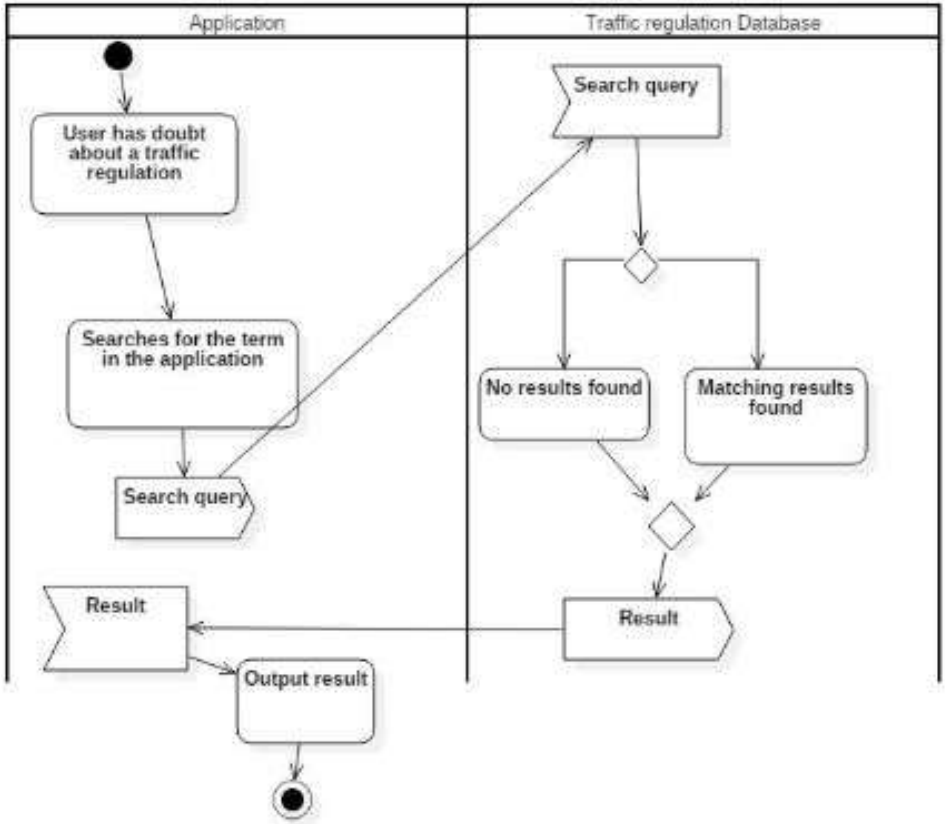
3. Construction process activity diagram

Here's another example of an activity diagram that is used to outline the steps of a business process. Utilizing different activity diagram symbols, this example makes use of a fork and join node to start and end a concurrent flow:



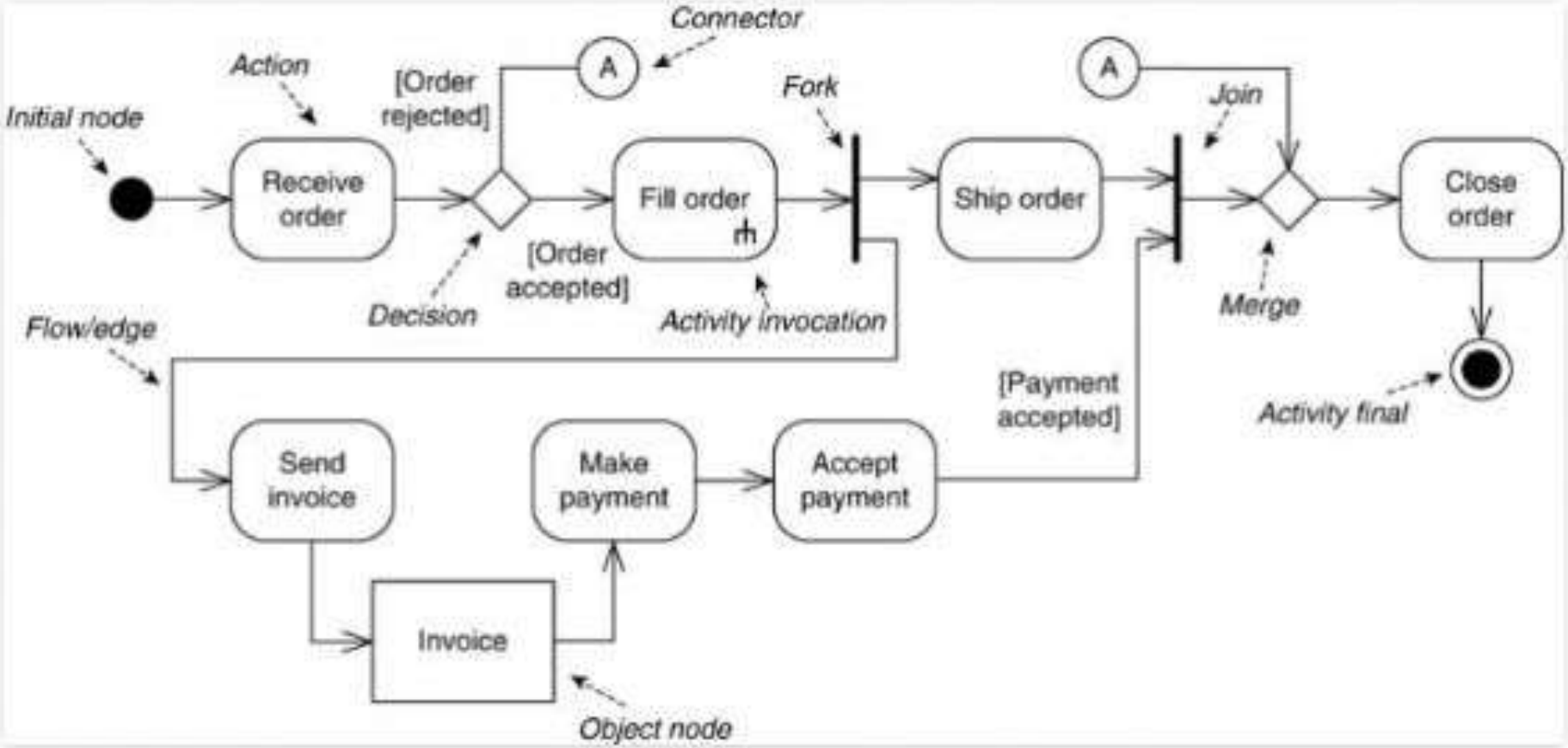
4. Traffic regulation search query process activity diagram

This activity diagram depicts the process of running a search query in a traffic regulation database. It features the sent and received signals as well as the object node to indicate that an object in the form of a search result is created from the last step in the process.



- **Start node**: A small filled circle that symbolizes the initial state or the start of the activity (initial node).
- **Activity nodes:** A rectangle with rounded corners that symbolizes an activity or action state (receive order, fill order, etc)
- **Action node**: A stadium or capsule-shaped symbol that is used to represent an action.
- **Action flow**: An arrow that represents the transition from one activity or action to another. Also called activity edge or control flow.
- **Object node:** A rectangle that represents an object that is created or used in the activity.
- **Object flow**: An elbow or dashed arrow that is placed after an action to show the creation of an object or before an action to show that it requires an object.
- **Fork node and join node**: Both are represented by a thick horizontal line that splits an action into concurrent flows (fork node) or joins concurrent flows into a single action (join node).
- **Time event**: An hourglass symbol that depicts a time interval within an activity.
- **Sent and received signals**: An arrow pentagon shape (signal sent) indicates that a signal needs to be received to complete an action, while a flag shape with a swallowtail (signal received) indicates that the signal has been received.
- **End node**: A small filled circle inside another circle that marks the final state or the end of the activity.

For your reference, some of them are included in this example:



Difference between an activity diagram and swimlanes:

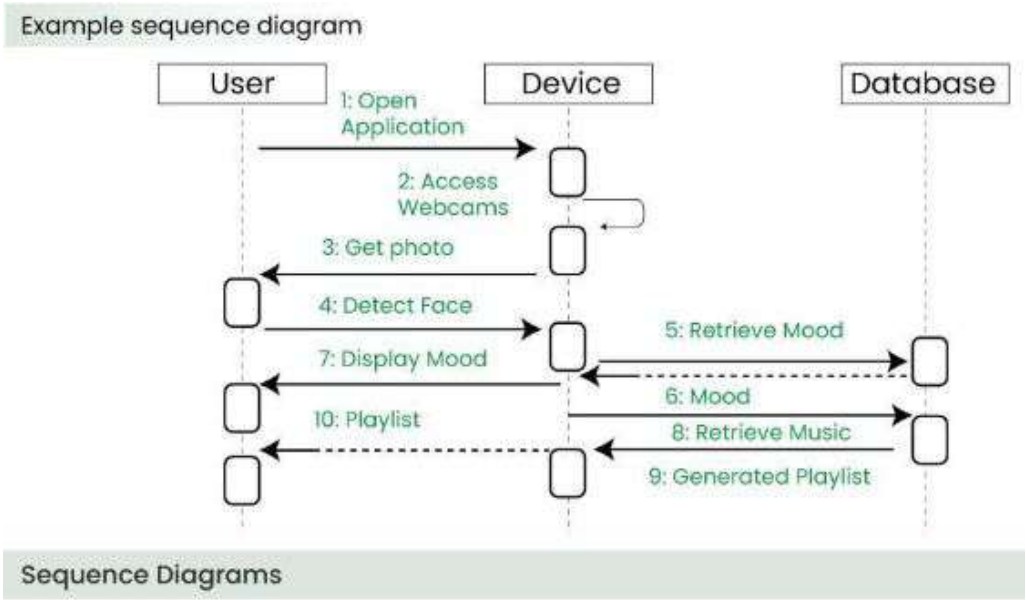
An activity diagram shows the flow of activities in a system while a swimlane diagram shows the individuals and departments responsible for the steps in a process.

Swimlanes only function as one part of an activity diagram and activity diagrams may or may not contain swimlanes.

Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.

- We can also use the terms event diagrams or event scenarios to refer to a sequence diagram.
- Sequence diagrams describe how and in what order the objects in a system function.
- These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.



Sequence Diagram Notations

Actors

An actor in a UML diagram represents a role interacting with the system and its objects.

It is always external to the system being modeled.

Actors include human users and external subjects, depicted using stick person notation.

Multiple actors can appear in a sequence diagram.

Lifelines

A lifeline represents an individual participant in a sequence diagram, with each instance being depicted by a lifeline.

Lifelines are displayed in a rectangle, called the head, containing their name and type.

The head connects to a vertical dashed line, or stem.

If the instance is unnamed, the lifeline's name is left blank.

Difference between a Lifeline and an Actor

Lifelines depict internal objects, while actors represent external objects.

Sequence Diagram Notations …

Messages

Messages illustrate communication between objects in sequential order along the lifeline and are shown using arrows.

Lifelines and messages are central to sequence diagrams.

Message Types

The following are the different message types –

Synchronous messages

A synchronous message requires a reply before the interaction can continue.

The sender waits until the receiver processes the message, proceeding only after receiving a reply.

Most object-oriented programming calls are synchronous, represented by a solid arrowhead.

Asynchronous Messages

Asynchronous messages do not wait for a reply.

The interaction moves forward whether or not the receiver processes the message.

A lined arrowhead represents this.

Sequence Diagram Notations ...

Create message

A Create message instantiates a new object in the sequence diagram.

When a message call requires creating an object, it's represented with a dotted arrow labeled "create."

Delete Message

A Delete Message removes an object.

When an object is deallocated or destroyed in the system, this is indicated by an arrow ending with an "x."

Self Message

When an object sends a message to itself, it is called a Self-Message, represented by a U-shaped arrow.

Reply Message

Reply messages show the receiver sending a message back to the sender. Represented by an open arrowhead with a dotted line, the interaction proceeds only after the receiver sends a reply.

Found Message

A Found message represents a situation where an unknown source sends a message. It is shown as an arrow directed at a lifeline from an endpoint.

Lost Message

A Lost message shows a scenario where the recipient is unknown. It is represented by an arrow from a lifeline directed at an endpoint.

Guards

Guards are used to model conditions in UML. They restrict the message flow based on whether a condition is met, helping developers understand the constraints of a system or process.

Purpose of Sequence Diagram

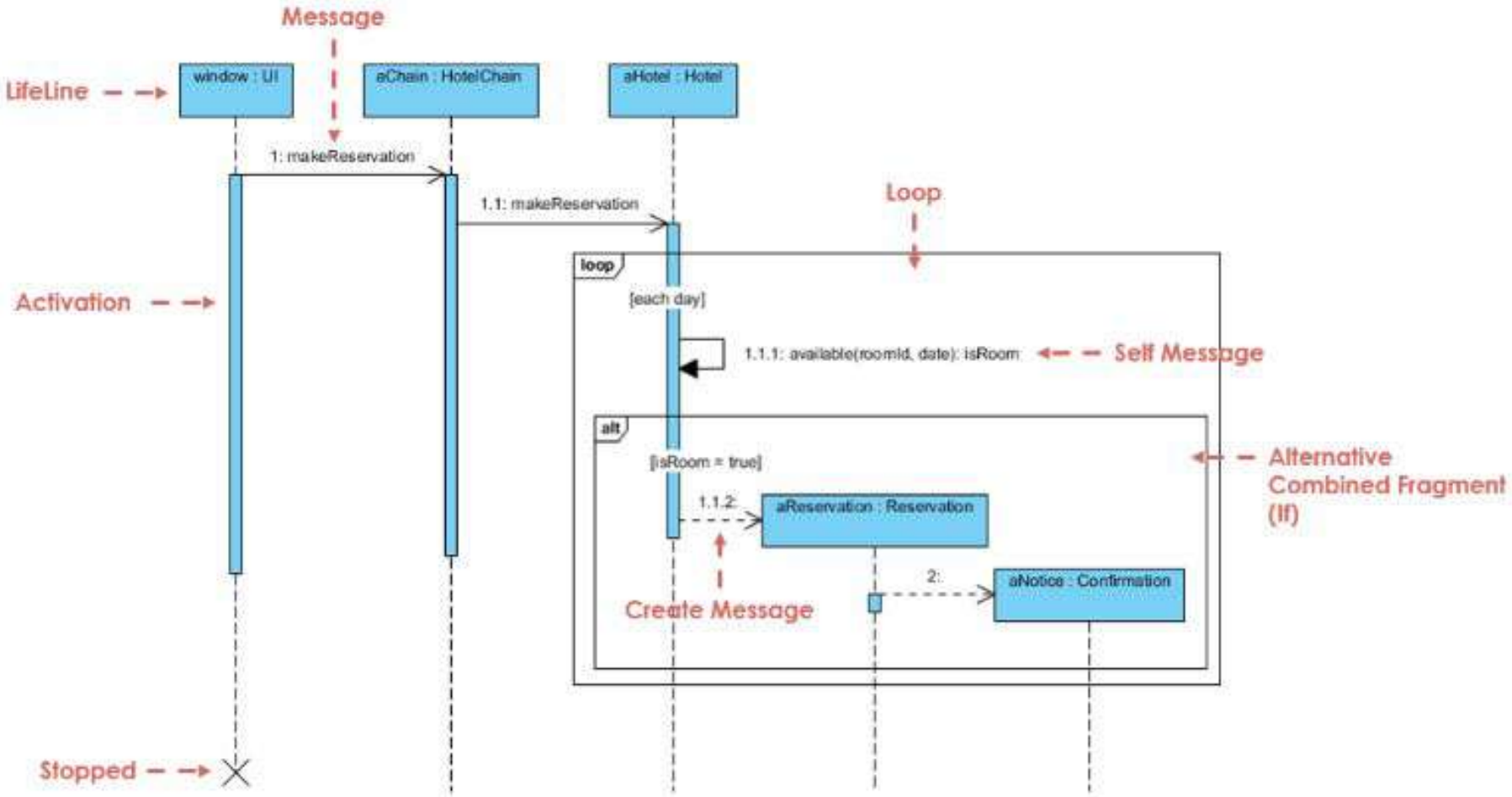
- Model high-level interaction between active objects in a system
- Model the interaction between object instances within a collaboration that realizes a use case
- Model the interaction between objects within a collaboration that realizes an operation
- Either model generic interactions (showing all possible paths through the interaction) or specific instances of a interaction (showing just one path through the interaction)

Weaknesses of Sequence Diagram

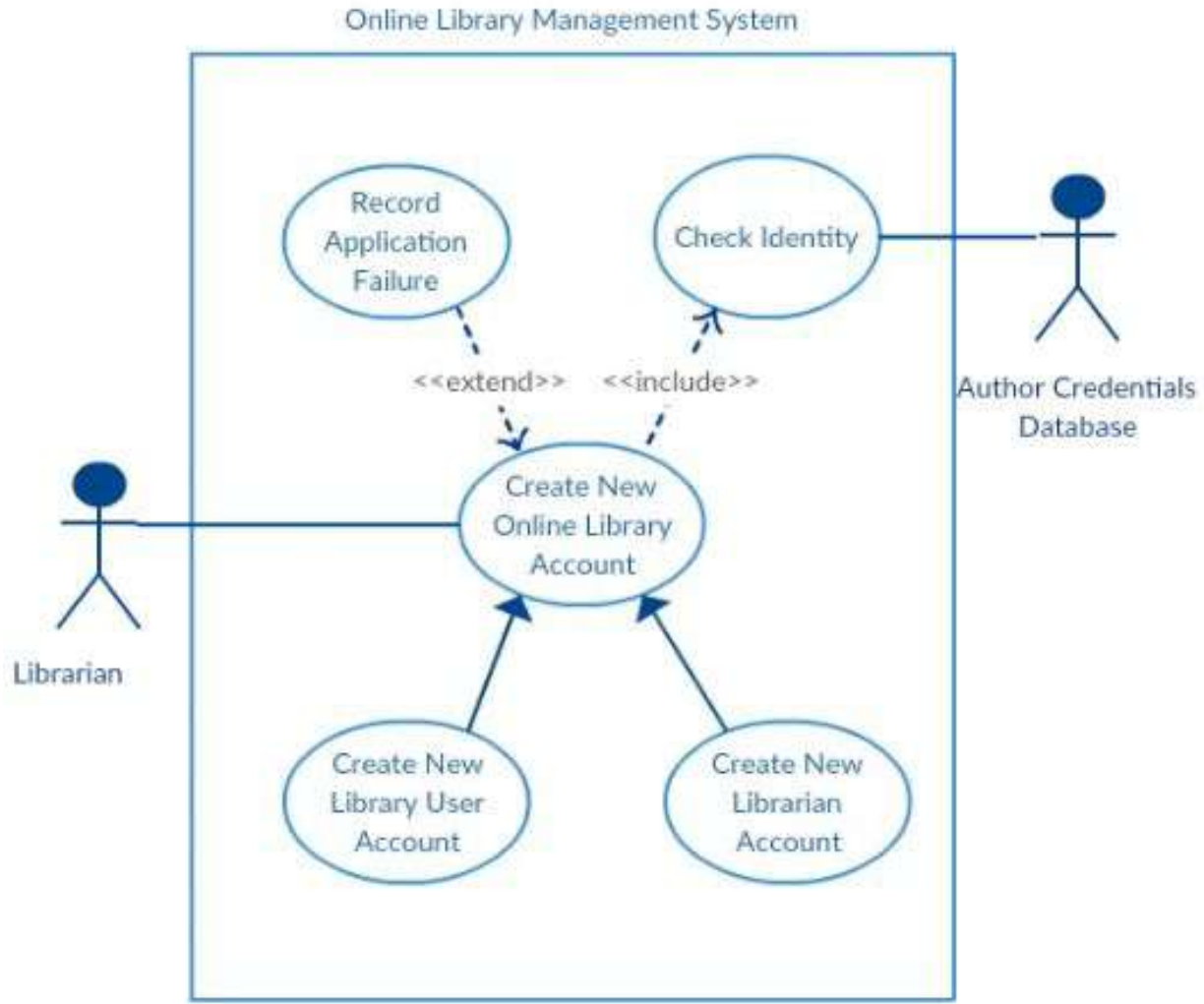
- Sequence **diagrams are not structural and cannot show any structural relationships** between classes.
- They can be used to identify missing classes, but they **cannot identify precise structural relationships** between them.
- While sequence diagrams show behavior in a temporal sequence, they are not pure dynamic diagrams as they **cannot show changes to the state of the object or changes to the values of the attributes of an object**.
- **It is difficult to show an if-then-else or a for-next condition on sequence diagrams.**
- Sequence diagrams may **become time-consuming to generate for complex systems** with many objects or interactions.
- They are **limited in handling unforeseen combinations** of abnormal events.
- Sequence diagrams **lack integration with established methods**, making them harder to use in some contexts.

Sequence Diagram Example: Hotel System

- Sequence Diagram is an interaction diagram that details how operations are carried out -- what messages are sent and when.
- Sequence diagrams are organized according to time.
- The time progresses as you go down the page.
- The objects involved in the operation are listed from left to right according to when they take part in the message sequence.
- Below is a sequence diagram for making a hotel reservation.
- The object initiating the sequence of messages is a Reservation window.



Note That: Class and object diagrams are static model views. Interaction diagrams are dynamic. They describe how objects collaborate.

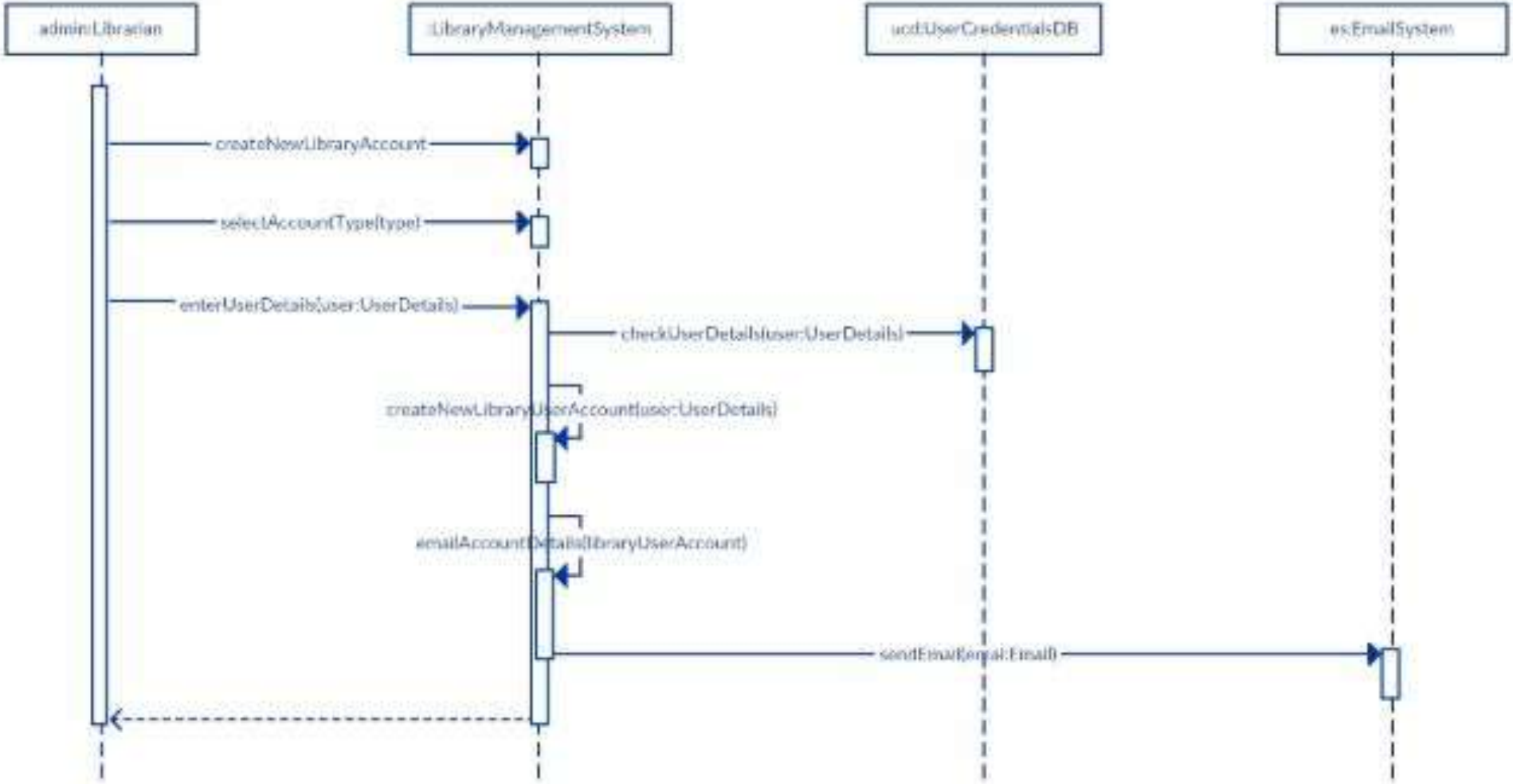

Online Library Management System
Record Application Failure
Check Identity
<<extend>>
<<include>>
Author Credentials Database
Create New Online Library Account
Librarian
Create New Library User Account
Create New Librarian Account

From the above use case diagram example of ‘Create New Online Library Account’, we will focus on the use case named ‘Create New User Account’ to draw our sequence diagram example.
Before drawing the sequence diagram, it’s necessary to identify the objects or actors that would be involved in creating a new user account.
These would be;

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

Once you identify the objects, it is then important to write a detailed description of what the use case does.
From this description, you can easily figure out the interactions (that should go in the sequence diagram) that would occur between the objects above, once the use case is executed.
Here are the steps that occur in the use case named ‘Create New Library User Account’.

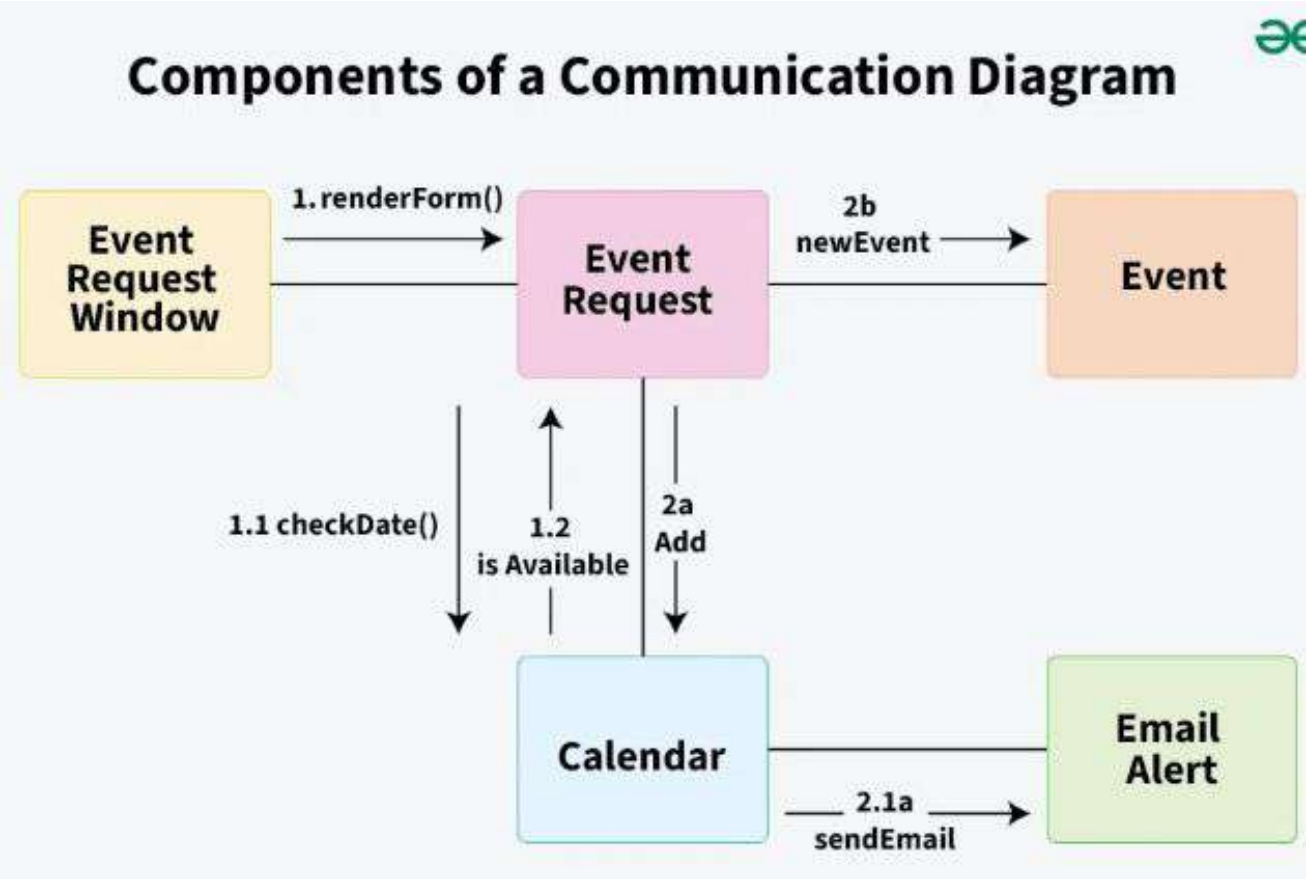
- The librarian request the system to create a new online library account
- The librarian then selects the library user account type
- The librarian enters the user’s details
- The user’s details are checked using the user Credentials Database
- The new library user account is created
- A summary of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


admin:Librarian
:LibraryManagementSystem
ucd:UserCredentialsDB
es:EmailSystem
createNewLibraryAccount
selectAccountType(type)
enterUserDetails(user:UserDetails)
checkUserDetails(user:UserDetails)
createNewLibraryUserAccount(user:UserDetails)
emailAccountDetails(libraryUserAccount)
sendEmail(email:Email)

Communication Diagram

A Communication Diagram (known as Collaboration Diagram in UML 1.x) is used to show sequenced messages exchanged between objects.

- A communication diagram focuses primarily on objects and their relationships.
- We can represent similar information using Sequence diagrams, however communication diagrams represent objects and links in a free form.



Collaboration Diagrams | Unified Modeling Language(UML)

- In UML (Unified Modeling Language), a Collaboration Diagram is a type of Interaction Diagram that **visualizes the interactions and relationships between objects in a system.**
- It shows **how objects collaborate to achieve a specific task or behavior**.
- Collaboration diagrams **are used to model the dynamic behavior of a system** and **illustrate the flow of messages between objects during a particular scenario or use case**.

What are Collaboration Diagrams?

- A collaboration diagram is a behavioral UML diagram which is also referred to as a communication diagram**.**
- It illustrates how objects or components interact with each other to achieve specific tasks or scenarios within a system.

In simpler terms, they visually represents the interactions between objects or components in a system and show how they collaborate to accomplish tasks within a system, and also illustrate the system's object architecture.

https://www.geeksforgeeks.org/system-design/collaboration-diagrams-unified-modeling-languageuml/

Importance of Collaboration Diagrams

Collaboration diagrams is important for understanding communication, design, analysis, and documentation of the system's architecture and behavior.

- **Visualizing Interactions:**
    - These diagrams offer a clear visual representation of how objects or components interact within a system.
    - This visualization helps stakeholders in understanding the flow of data and control for easier understanding.
- **Understanding System Behavior:**
    - By showing interactions, collaboration diagrams provide insights into the system's dynamic behavior during operation.
    - Understanding this behavior is important for identifying potential issues, optimizing performance, and ensuring the system functions smoothly.
- **Facilitating Communication:**
    - Collaboration diagrams provide an effective communication tools among team members.
    - They facilitate discussions, enabling refinement of the system's design, architecture, and functionality.

Components and their Notations in Collaboration Diagrams

There are several components in Collaboration Diagram. Let's see those components and their notations:

1. Objects/Participants

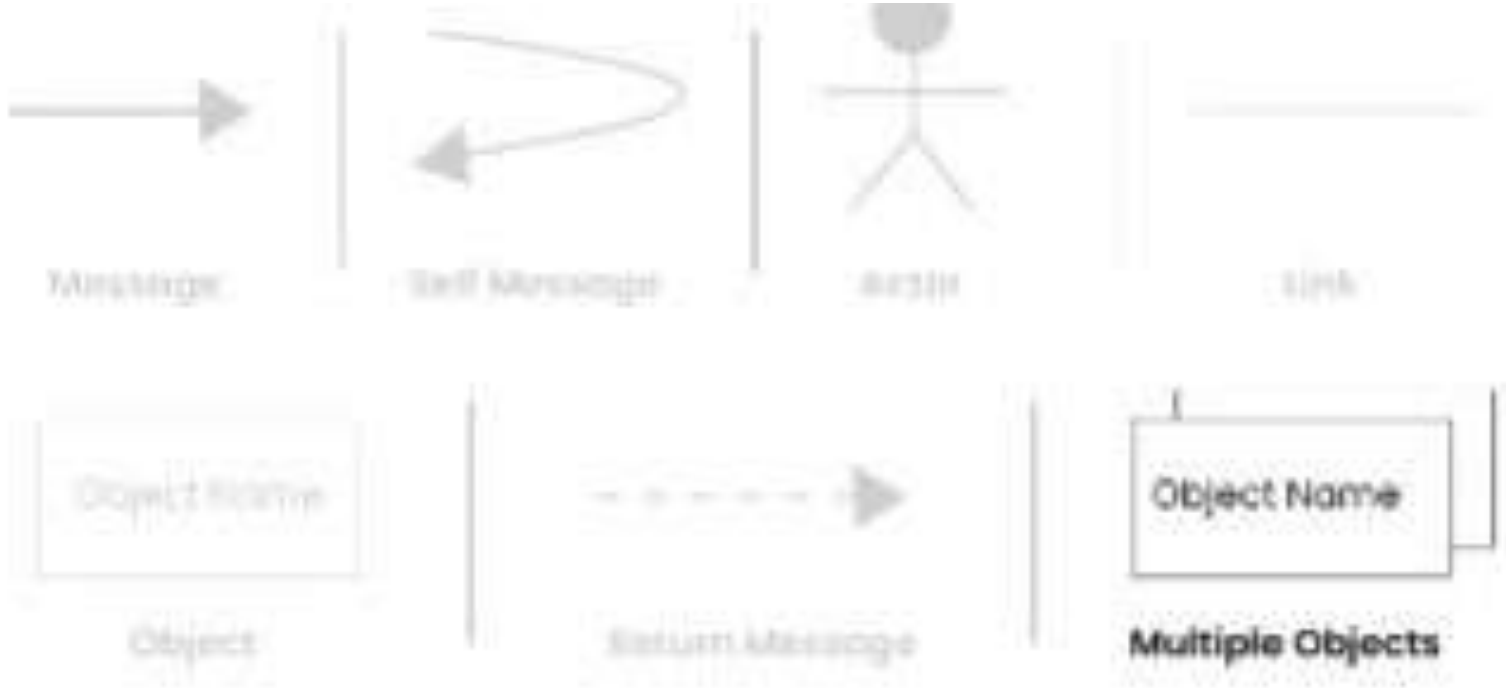
Objects are represented by rectangles with the object's name at the top.

Each object participating in the interaction is shown as a separate rectangle in the diagram.

Objects are connected by lines to indicate messages being passed between them.

2. Multiple Objects

Multiple objects are represented by rectangles,each with the object's name inside, and interactions between them are shown using arrows to indicate message flows.

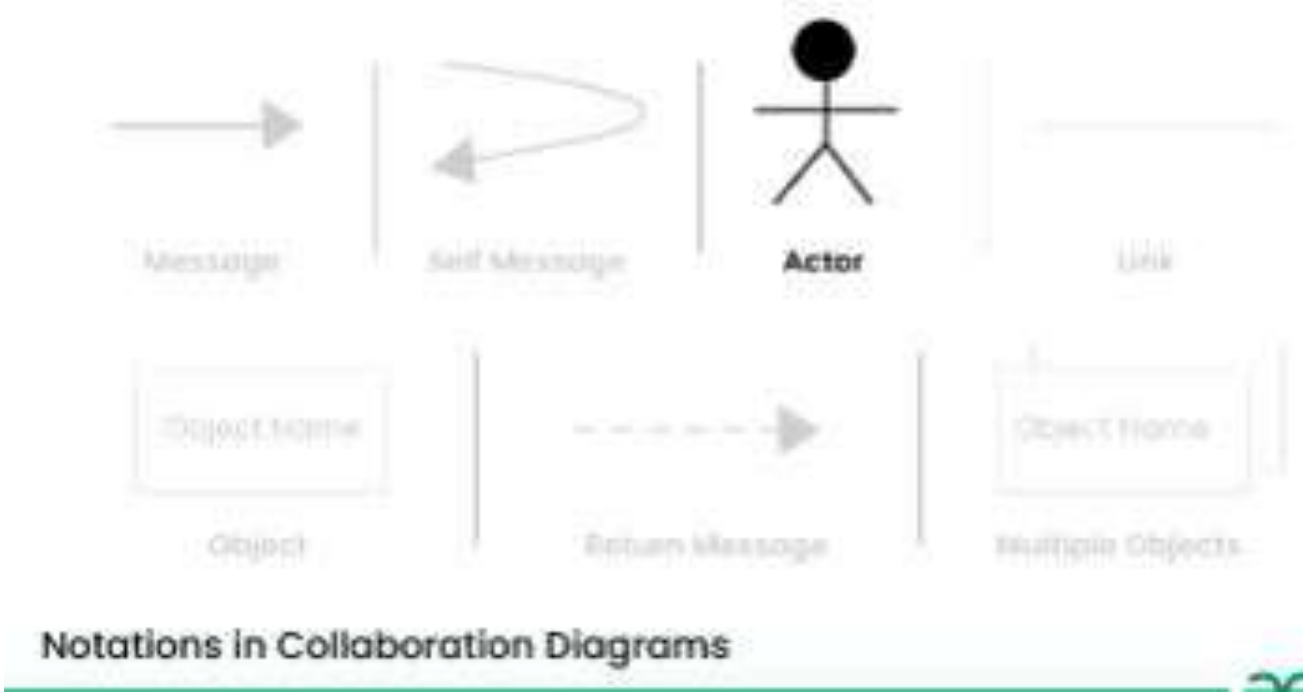


3. Actors

They are usually shown at the top or side of the diagram.

Actors indicate their involvement in the interactions with the system's objects or components.

They are connected to objects through messages, showing the communication with the system.



4. Messages

Messages represent communication between objects.

Messages are shown as arrows between objects, indicating the flow of communication.

Each message may include a label indicating the type of message (e.g., method call, signal).

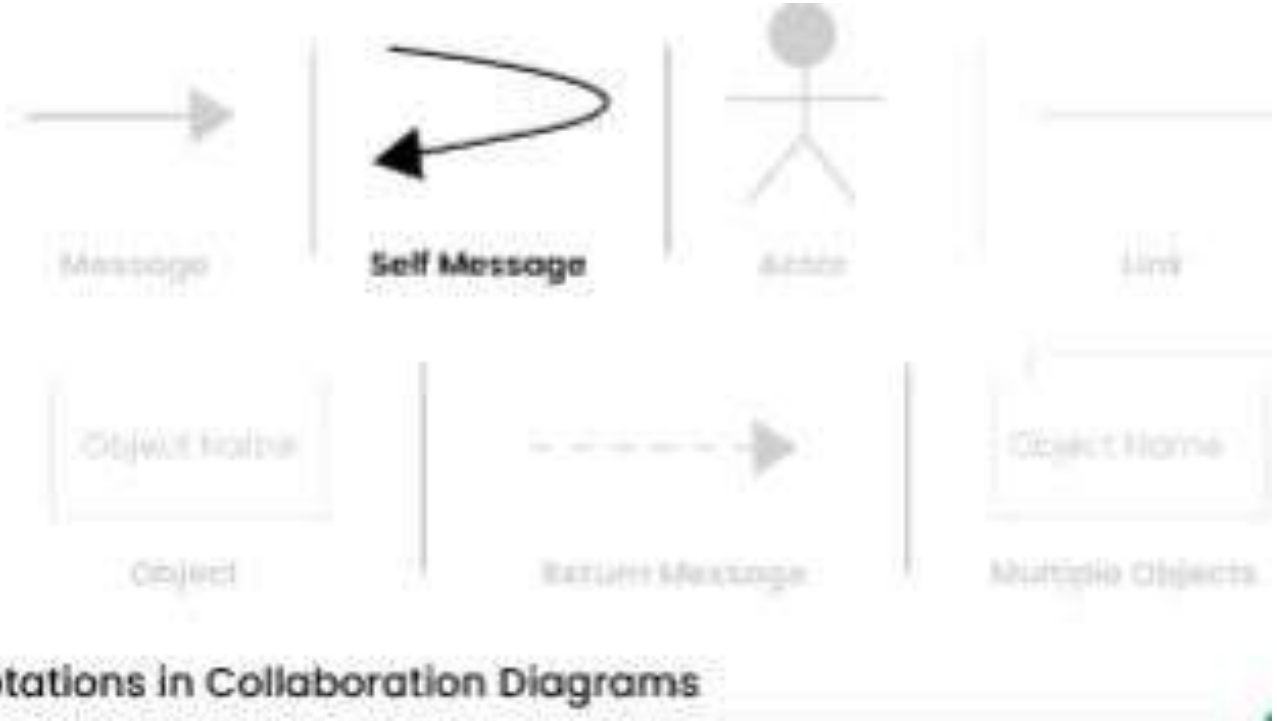
Messages can be **asynchronous** (indicated by a dashed arrow) or **synchronous** (solid arrow).

5. Self Message

This is a message that an object sends to itself.

It represents an action or behavior that the object performs internally without involving any other objects.

Self-messages are useful for modeling scenarios where an object triggers its own methods or processes.

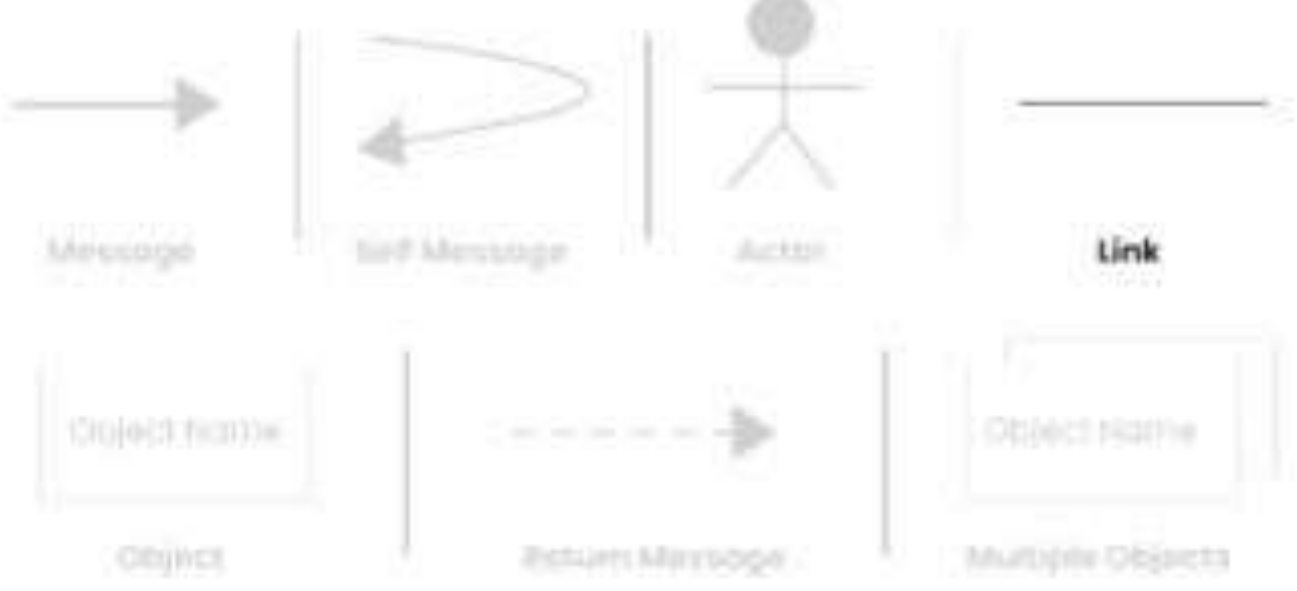


6. Links

Links represent associations or relationships between objects.

Links are shown as lines connecting objects, with optional labels to indicate the nature of the relationship.

Links can be uni-directional or bi-directional, depending on the nature of the association.



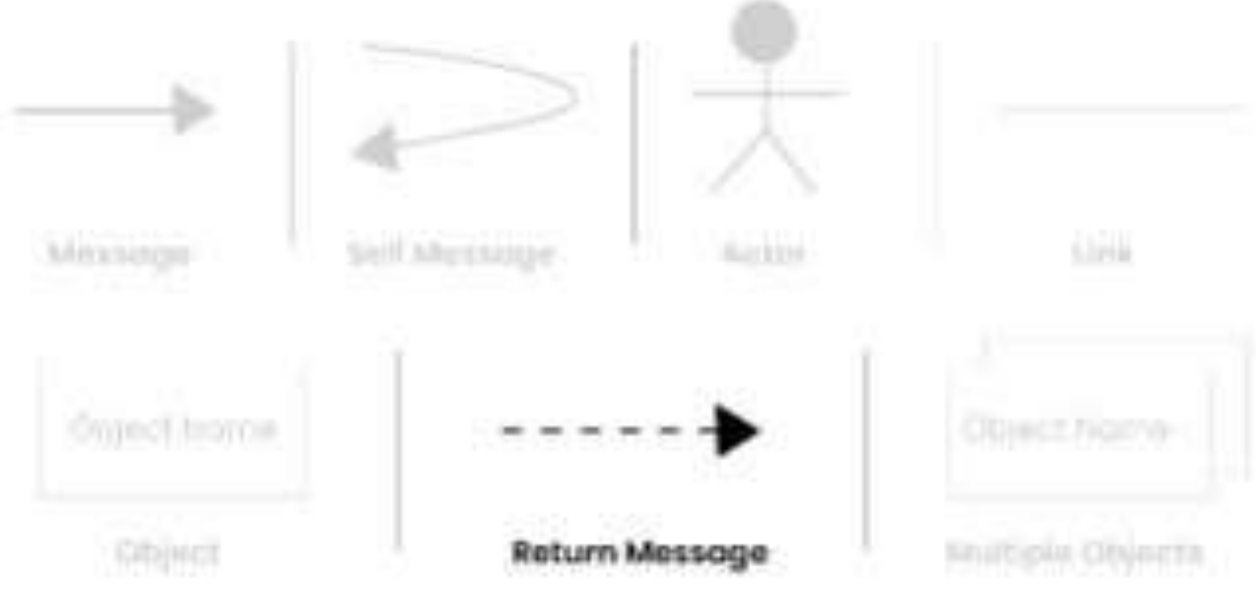
Notations in Collaboration Diagrams

7. Return Messages

Return messages represent the return value of a message.

They are shown as dashed arrows with a label indicating the return value.

Return messages are used to indicate that a message has been processed and a response is being sent back to the calling object.



How to draw Collaboration Diagrams?

Below are the steps to draw collaboration diagrams:

- **Step 1: Identify Objects/Participants:** Start by figuring out the objects or participants in the system. These can be classes, modules, actors, or any other important entities.
- **Step 2: Define Interactions:** Determine how these objects work together to complete tasks or scenarios in the system. Identify the messages they exchange during these interactions.
- **Step 3: Add Messages:** Draw arrows between lifelines to show the messages exchanged between objects. Label each arrow with the message name and any relevant parameters or data being sent.
- **Step 4: Consider Relationships:** If there are connections or dependencies between objects, show these using the right notations, like dashed lines or arrows.
- **Step 5: Documentation:** Once you're done, document the collaboration diagram with any necessary explanations or notes. Make sure the diagram clearly communicates the system's interactions to stakeholders

Use cases of Collaboration Diagrams

Below are the main use cases of collaboration diagrams:

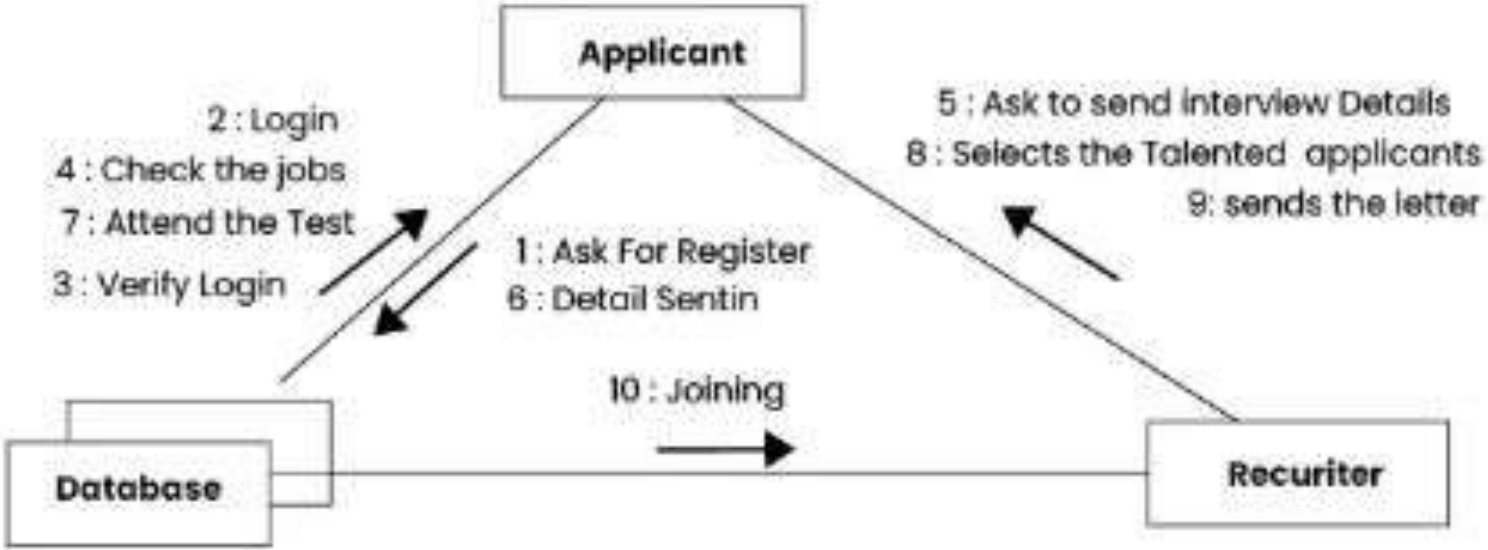
- Collaboration diagrams **help developers see how various parts of a system work together, making it easier to build and test software.**
- They **assist in visualizing interactions within the system, helping analysts and designers improve the system's architecture.**
- They are **crucial for documenting system architecture and design choices**, serving as useful reference materials for developers and testers.
- Collaboration diagrams **help trace message flow and spot system issues**, helping in debugging and troubleshooting tasks

Real-World Example of Collaboration Diagram

Let's understand collaboration diagram using the example of Job Recruitment System.

The recruiter object interacts with the database object to verify the login, check the jobs, select a talented applicant, and send interview details.

- The applicant object interacts with the database object to provide details and attend the test.
- The collaboration diagram shows the sequential order of these interactions and the relationship between the objects involved.



Job Recruitment system

1. Applicant

This object represents a job candidate applying for a position.

The Applicant interacts with the Recruiter to share personal and professional information and participate in the interview.

After the interview, the Recruiter selects the promising candidate and sends a joining letter to the Applicant.

Applicant --attend test --> Database

Applicant --provide details --> Database

2. Recruiter

- This object represents the person or system responsible for hiring new employees.
- The Recruiter interacts with the Applicant to verify login details, check available job positions, select suitable candidates, send interview information, and issue joining letters.
- The Recruiter also works with the Database to retrieve and update necessary information.
- ***Recruiter --verify login--> Database***
- ***Recruiter <-- confirms login --> Database***
- ***Recruiter --check jobs positions --> Database***
- ***Recruiter --select talented applicant --> Applicant***
- ***Recruiter --send interview details --> Applicant***
- ***Recruiter --send joining letter --> Applicant***

3. Database

- This object represents the system that stores important information for the recruitment process.
- The Database interacts with the Recruiter to provide job positions, verify logins, and send details about applicants.
- It also interacts with the Applicant to store and retrieve necessary information about them.
- ***Database --send jobs --> Recruiter***

Benefits of Collaboration Diagrams

- Below are the benefits of collaboration diagram:
  - Collaboration diagrams simplify how system components interact, minimizing confusion among team members.
  - They enhance discussions and decision-making by offering a visual representation of system interactions that everyone can understand.
  - Collaboration diagrams help visualize data and control flow within the system that support system's analysis, design, and documentation.
  - They help in debugging by showing the sequence of interactions and potential error sources.
  - By making streamlined development and reducing misunderstandings, collaboration diagrams enhance overall efficiency in system development and maintenance.

Challenges of Collaboration Diagrams

- Below are the challenges of Collaboration Diagram:
  - Keeping collaboration diagrams clear in large systems with many objects and interactions can be tough.
  - Sometimes, interpreting the interactions in diagrams isn't straight forward, leading to misunderstandings.
  - Diagrams might not fully capture systems where interactions change over time, like those with continuously processing and improvising.
  - Ensuring that diagrams effectively convey complex interactions to all stakeholders can be challenging.