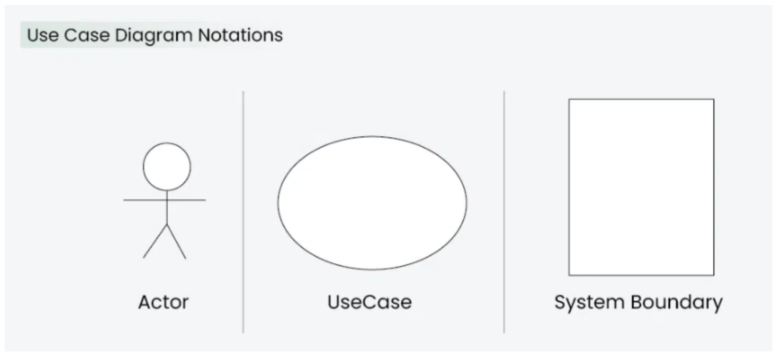
System Modeling

Notes by Mannan Ul Haq

Use Case Diagrams

A Use Case Diagram is a vital tool in system design, it provides a visual representation of how users interact with a system. It serves as a blueprint for understanding the functional requirements of a system from a user's perspective.

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.



Actors

Actors are external entities that interact with the system.

Use Cases

Use cases are like scenes in the play. They represent specific things your system can do.

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.

Use Case Diagram Relationships

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.

Include Relationship:

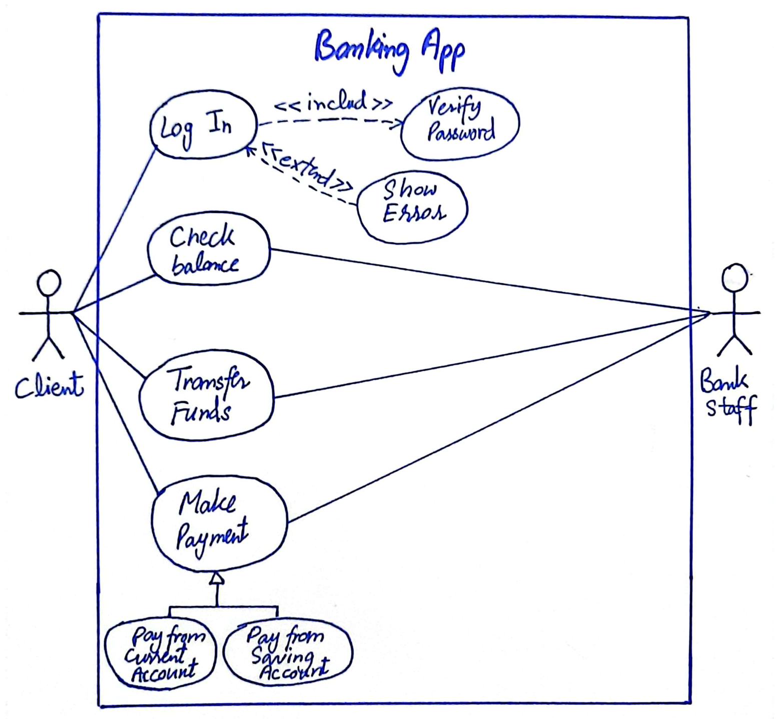
The Include Relationship indicates that a use case includes the functionality of another use case. It is denoted by a dashed arrow pointing from the including use case to the included use case.

Extend Relationship:

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

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.



Sequence Diagram

Lifelines:

A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram.

Messages:

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

A sequence diagram simply depicts the interaction between the objects in a sequential order i.e. the order in which these interactions occur. Sequence diagrams describe how and in what order the objects in a system function.

Messages can be broadly classified into the following categories:

1. **Synchronous messages:**

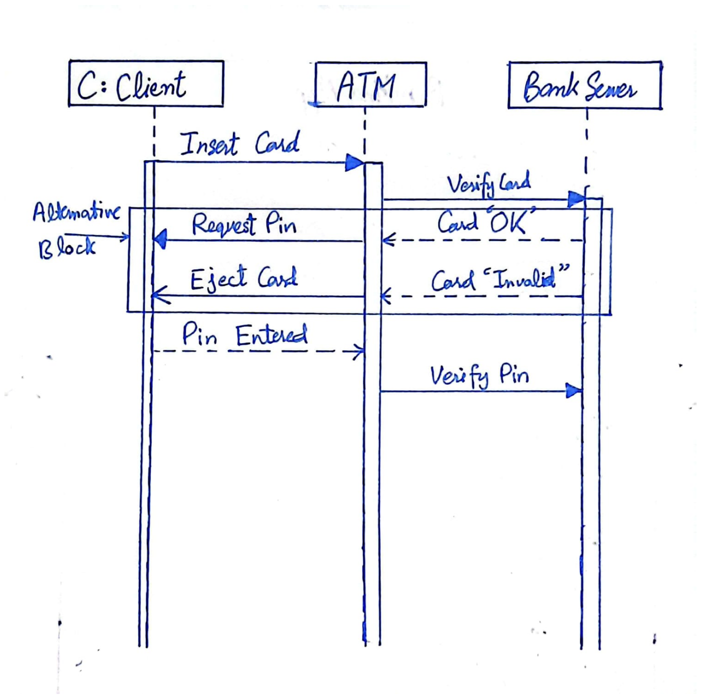
A synchronous message waits for a reply before the interaction can move forward. We use a **solid arrow head** to represent a synchronous message.

2. **Asynchronous Messages**

An asynchronous message does not wait for a reply from the receiver. We use a **lined arrow head** to represent an asynchronous message.

3. **Reply Message:**

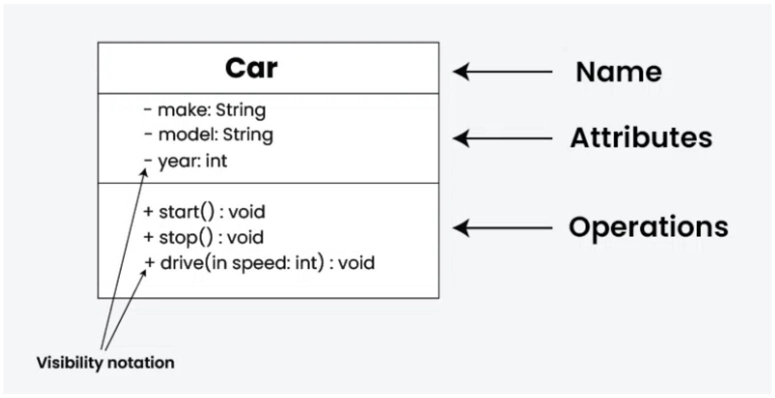
Reply messages are used to show the message being sent from the receiver to the sender. We represent a return/reply message using an **open arrow head with a dotted line**.



Class Diagram

Class diagrams are a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes within a system i.e. used to construct and visualize object-oriented systems.

Notations:



1. **Class Name:**

   The name of the class is typically written in the top compartment of the class box and is a noun.

2. **Attributes:**

   Attributes, also known as properties or fields, represent the data members of the class.

3. **Methods:**

   Methods, also known as functions or operations, represent the behavior or functionality of the class.

4. **Visibility Notation:**

   Visibility notations indicate the access level of attributes and methods. Common visibility notations include:

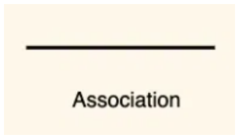
   - **+** for public (visible to all classes)
   - **-** for private (visible only within the class)
   - **#** for protected (visible to subclasses)

Relationships between classes:

In class diagrams, relationships between classes describe how classes are connected or interact with each other within a system.

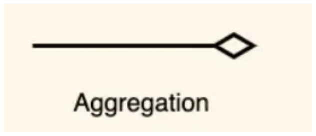
1. Association

An association represents a bi-directional relationship between two classes. It indicates that instances of one class are connected to instances of another class.



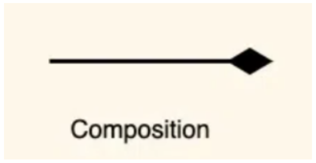
2. **Aggregation**

Aggregation is a specialized form of association that represents a “whole-part” relationship. It denotes a stronger relationship where one class (the whole) contains or is composed of another class (the part). Aggregation is represented by a diamond shape on the side of the whole class. In this kind of relationship, the child class can exist independently of its parent class.



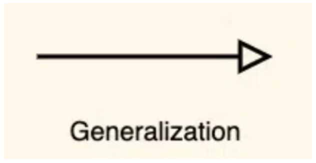
3. **Composition**

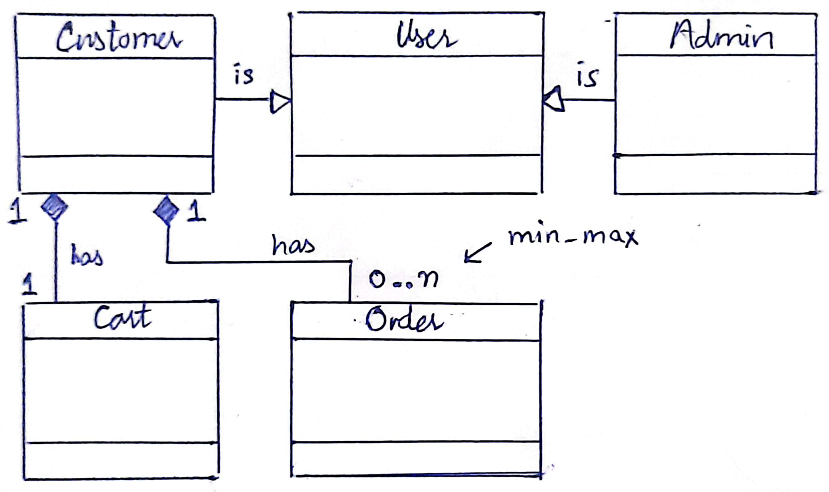
Composition is a stronger form of aggregation, indicating a more significant ownership or dependency relationship. In composition, the part class cannot exist independently of the whole class. Composition is represented by a filled diamond shape on the side of the whole class.



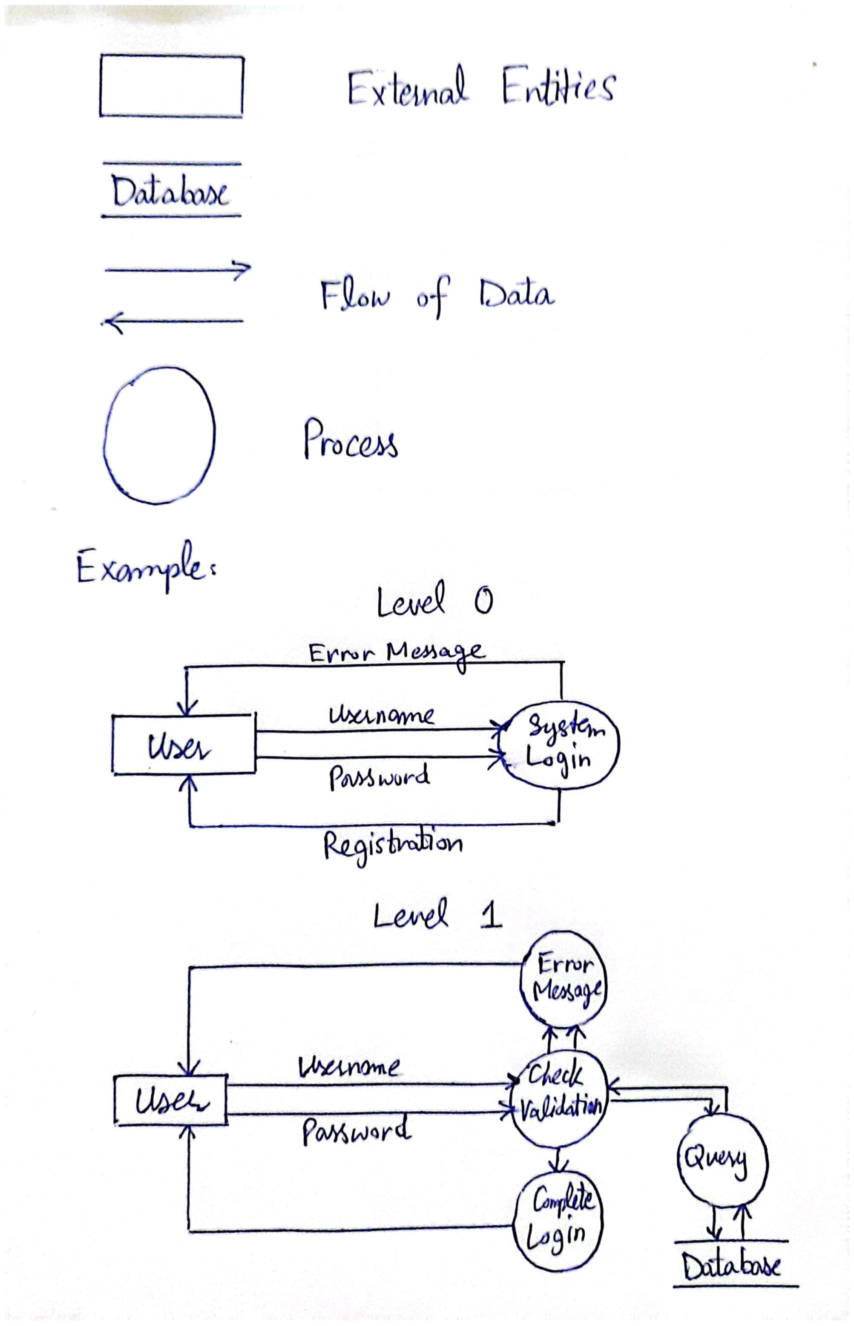
5. **Generalization(Inheritance)**

Inheritance represents an "is-a" relationship between classes, where one class (the subclass or child) inherits the properties and behaviors of another class (the superclass or parent). Inheritance is depicted by a solid line with a closed, hollow arrowhead pointing from the subclass to the superclass.





Data Flow Diagram:



Activity Diagram:

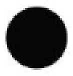
Activity Diagrams are used to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case.

An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.

Notations:

1. **Initial State:**

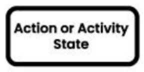
The starting state before an activity takes place is depicted using the initial state. We use a black filled circle to depict the initial state of a system.



Initial State

2. **Action or Activity State:**

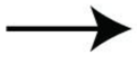
An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.



Activity State

3. **Action Flow or Control flows:**

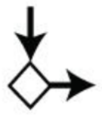
Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another activity state. An activity state can have multiple incoming and outgoing action flows.



Control Flow

4. **Decision node and Branching:**

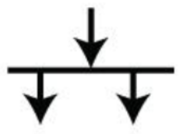
When we need to make a decision before deciding the flow of control, we use the decision node.



Decision node

5. **Fork:**

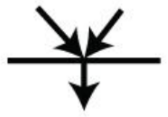
Fork nodes are used to support concurrent activities. When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement.



Fork

6. **Join:**

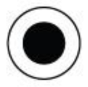
Join nodes are used to support concurrent activities converging into one.



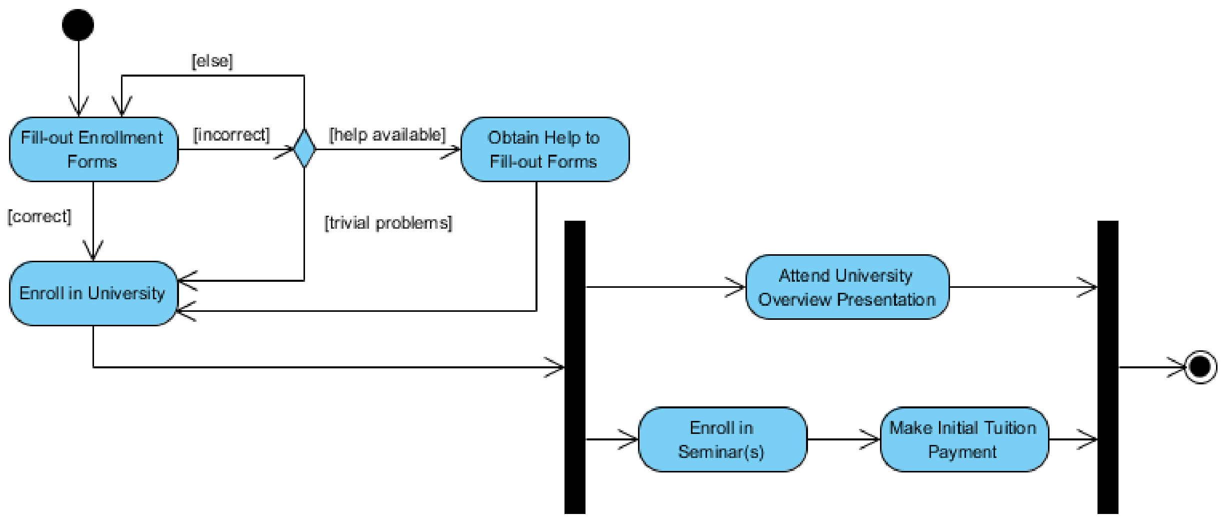
Join

7. **Final State or End State:**

The state which the system reaches when a particular process or activity ends is known as a Final State or End State.



Final State


[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