System Study, Feasibility Study, Normalization, UML Diagrams, Data Sanitization & Indexing

Ms. Navyamol K T, Asst. Professor, AJCE

Scenario: A local coffee shop owner wants to develop a mobile app to allow customers to place orders for pickup and delivery.

- **SYSTEM STUDY**

1. **Explaining the System: counter, kitchen, seating area**
2. **Understanding User Needs**
3. **Gathering Requirements**
4. **Creating Use Cases: choosing items, adding to cart, checkout, and receiving the order.**
5. **User Personas: busy professionals, students, coffee enthusiasts**
6. **Challenge of Poor System Study: orders being lost, incorrect items, or frustrated customers**

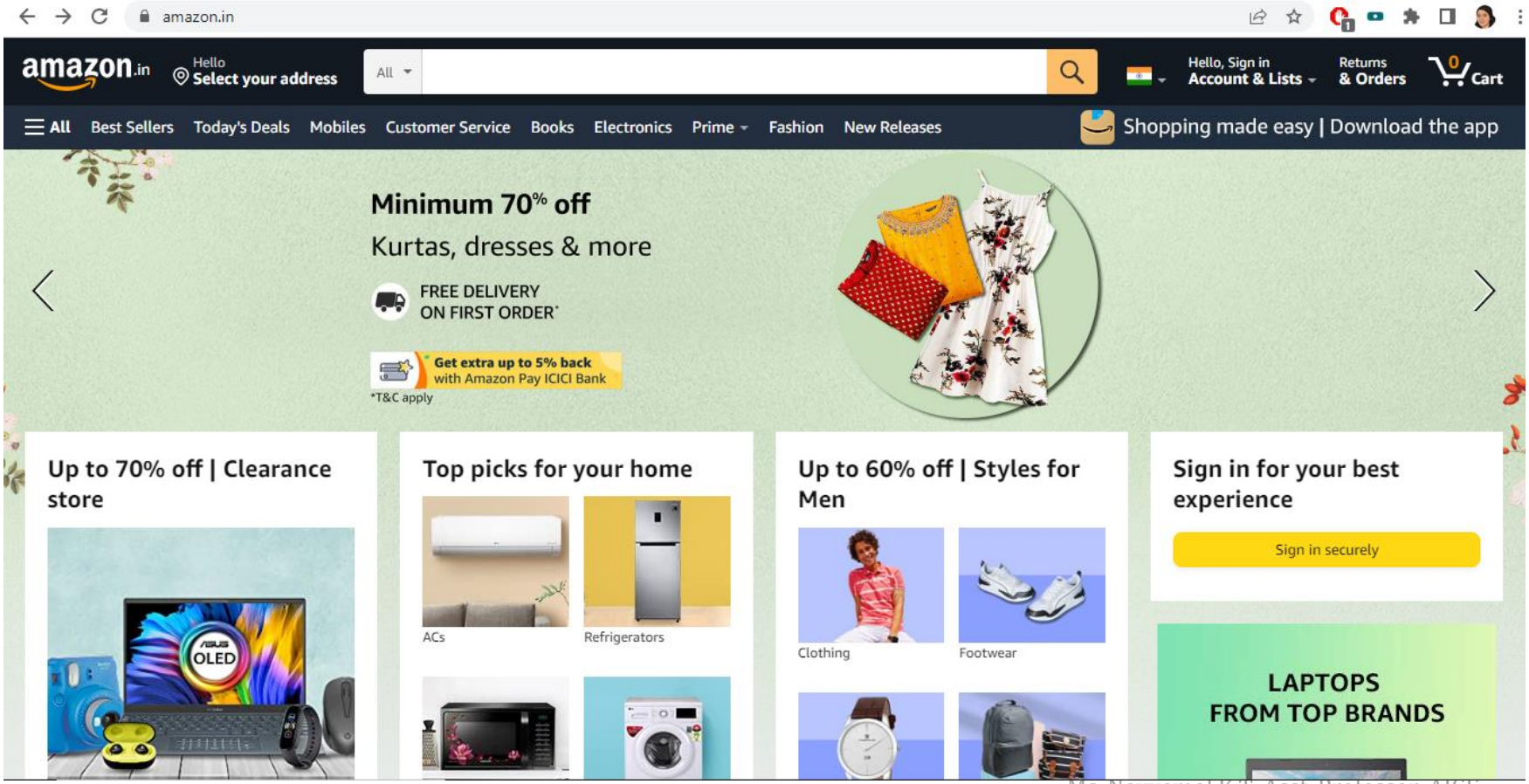

Coffee Business Application Development
System
267 ml
13 g
Foam
Coffee
Milk
119
67
81
My Cappuccino
CANCEL
SAVE

System Study

Natural System

Identify the natural or physical system that corresponds to your project topic.

Designed System



Instruction for System Study

1. **Selection of Project Topic:**

- Choose a topic for your project. For example, if your project is an online shopping site, the topic can be "Online Shopping Platform."

2. **Understanding Physical Systems:**

- Identify the natural or physical system that corresponds to your project topic.
- For an online shopping platform, the physical system is a traditional physical store.
- Visit a local store or observe a physical store's operations, noting key elements such as inventory management, customer interactions, sales processes, and payment methods.
- Document your observations in detail. Highlight strengths, weaknesses, and areas for improvement.

Instruction for System Study

3. Analyzing Designed Systems:

- Research and identify 5-10 existing websites or digital solutions that serve the same purpose as your project topic.
- For an online shopping platform, examples of designed systems include Amazon.in, Flipkart, Myntra, Snapdeal, and eBay.
- Analyze these websites based on the following criteria:
  - User interface and user experience (UI/UX)
  - Features and functionalities (e.g., product search, shopping cart, payment options)
  - Performance and reliability
  - Security measures
  - Customer reviews and feedback
  - Technological stack (if available)
  - Take detailed notes on each criterion for every website.

Instruction for System Study

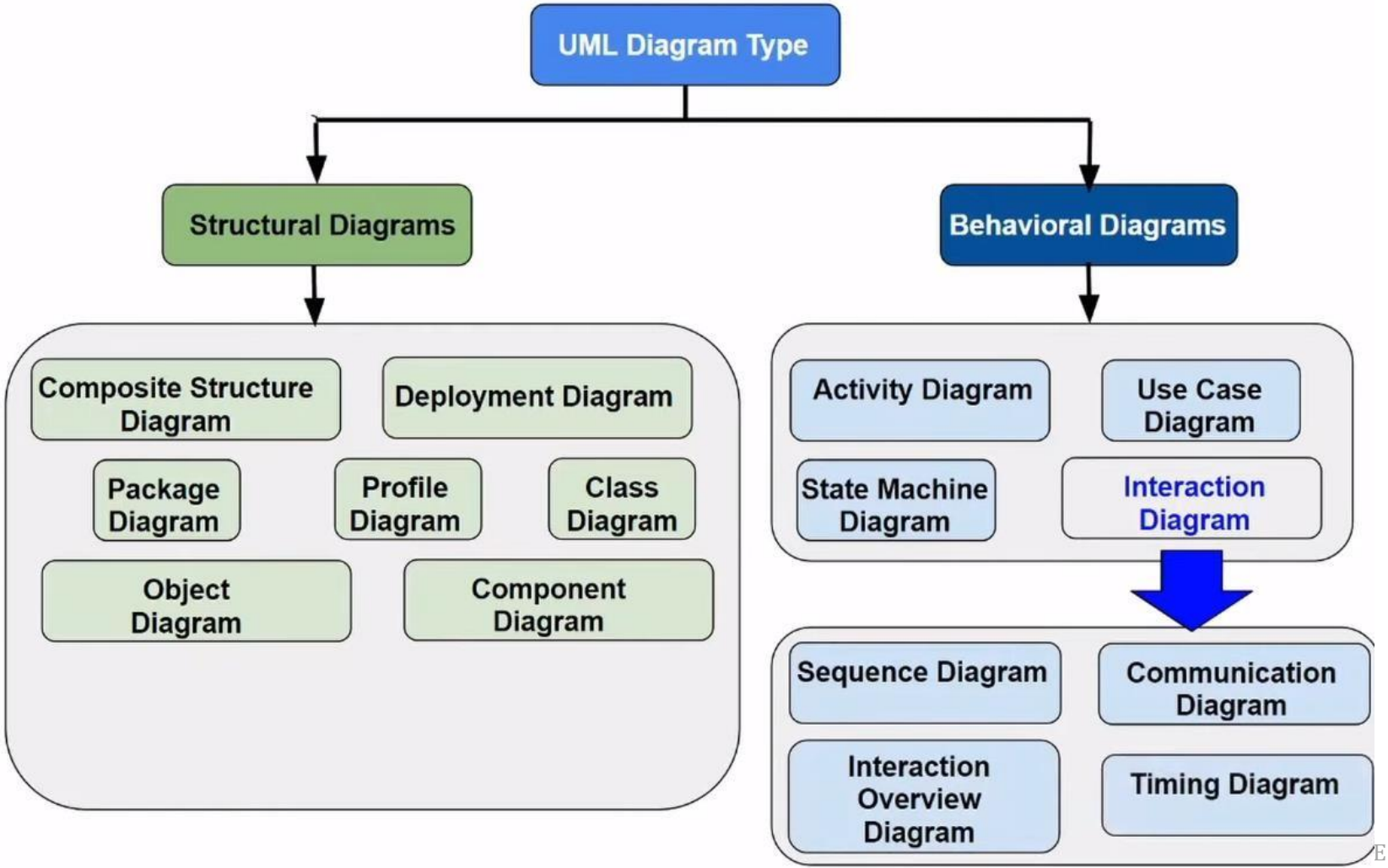
4. Comparison and Analysis:

- Compare the physical system (traditional store) with the designed systems (websites).
- Identify the pros and cons of each approach.
- Consider how the designed systems address or fail to address the limitations of the physical system.
- Summarize your findings in a comparative analysis report.

5. Documentation:

- Prepare a comprehensive report including:
- Introduction to your project topic
- Detailed observations of the physical system
- Analysis of the designed systems with specific examples
- Comparative analysis highlighting key insights and takeaways
- Conclusion with recommendations for your project based on the study

UNIFIED MODELING LANGUAGE™


UML Diagram Type
Structural Diagrams
Behavioral Diagrams
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Class Diagram
Object Diagram
Component Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

UML Standard Diagrams

Structural Diagrams

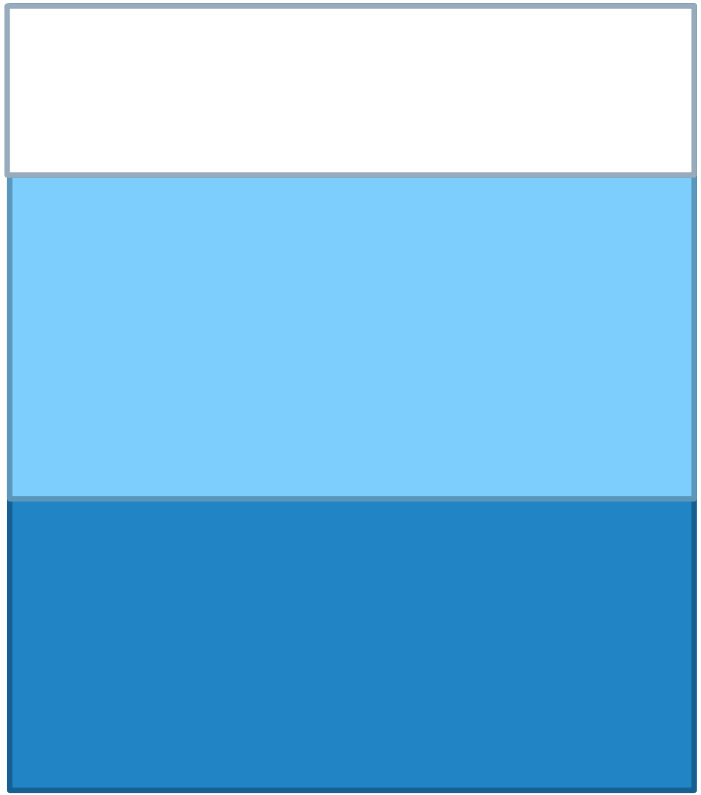
- **Behavioural Diagrams**

Structural Diagrams

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram

Class Diagram

Class Block



Vital components of a Class Diagram

Upper Section: The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken into account while representing a class are given below:

- Capitalize the initial letter of the class name.
- Place the class name in the center of the upper section.
- A class name must be written in bold format.
- The name of the abstract class should be written in italics format.

Vital components of a Class Diagram

Middle Section: The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:

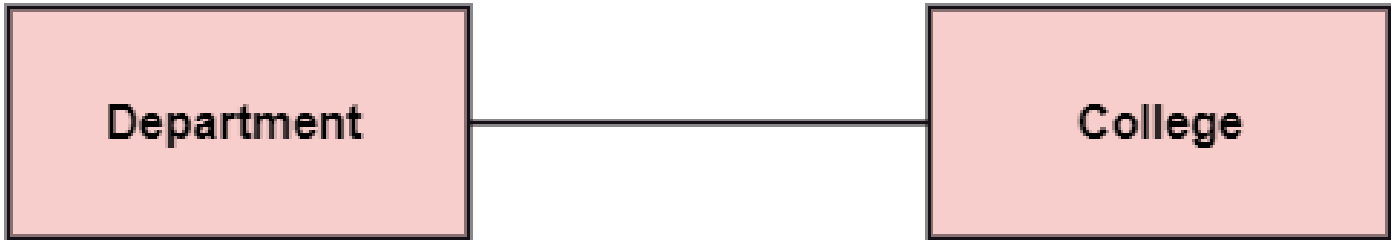
- The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).
- The accessibility of an attribute class is illustrated by the visibility factors.
- A meaningful name should be assigned to the attribute, which will explain its usage inside the class.

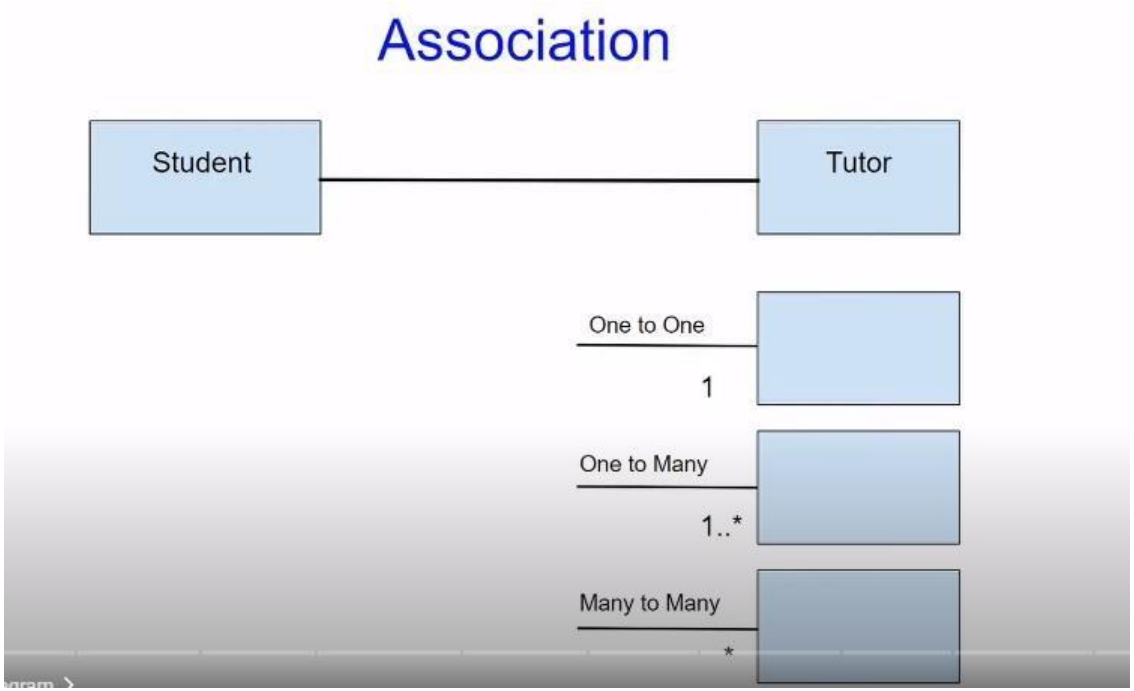
Lower Section: The lower section contain methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.

Relationships between classes in UML

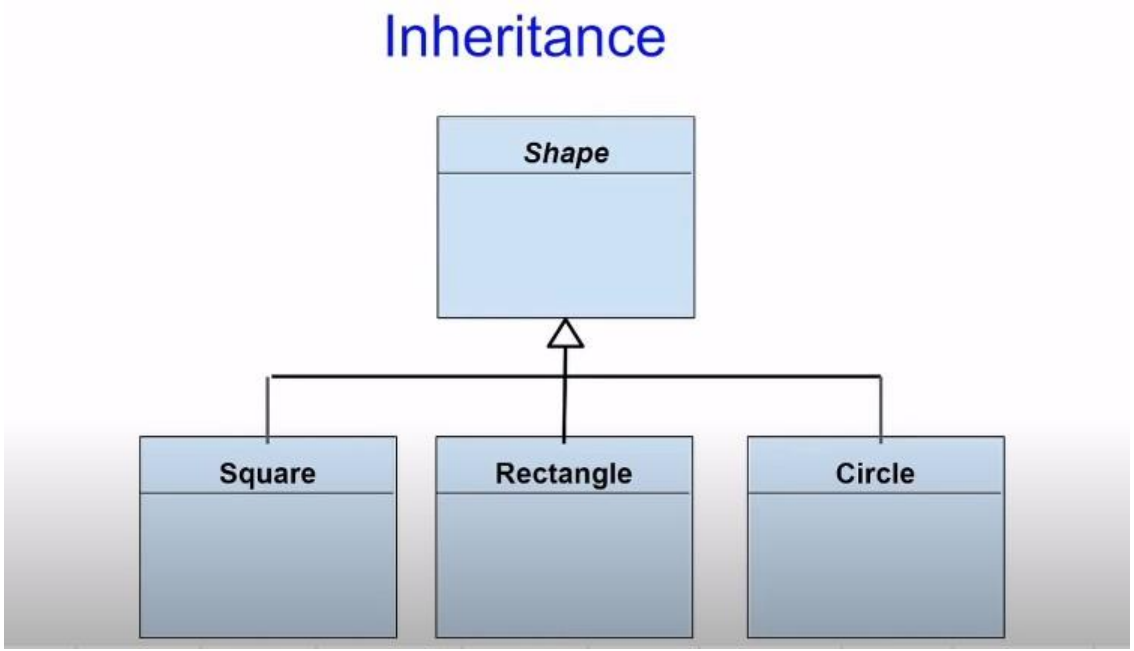
Association

- Association: It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship.
- For example, a department is associated with the college.




Association
Student
Tutor
One to One
1
One to Many
1..*
Many to Many
*

Inheritance

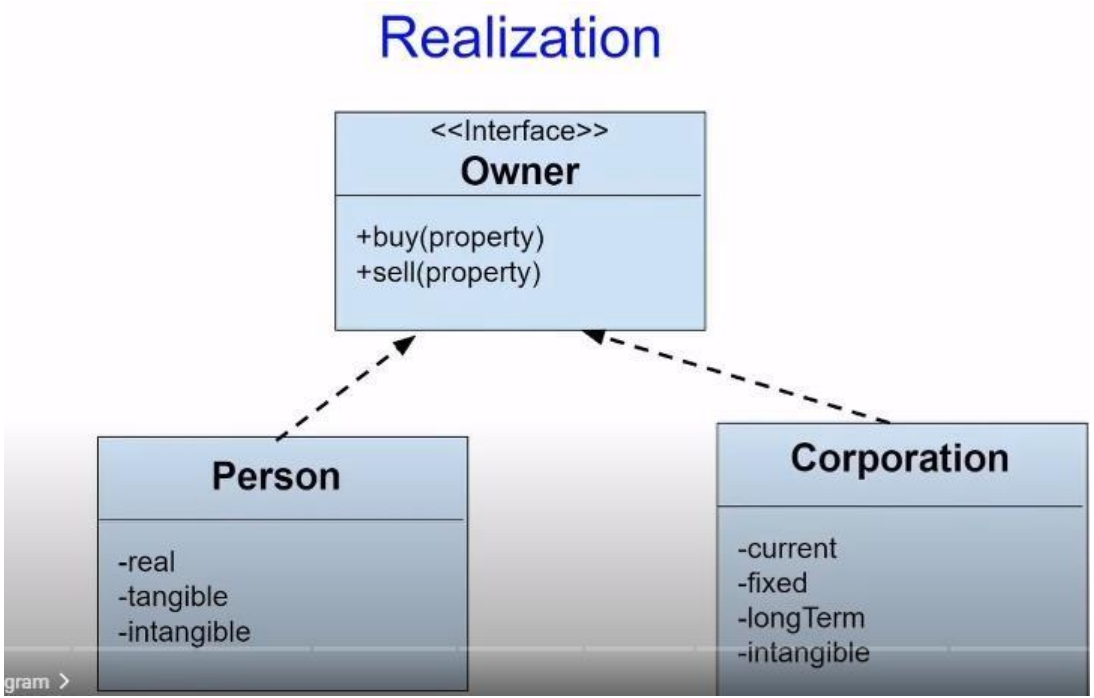


Aggregation

- Aggregation: An aggregation is a subset of association, which represents has a relationship. It is more specific then association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class.

- Composition: The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship.

Realization



Instructions to draw Class Diagram

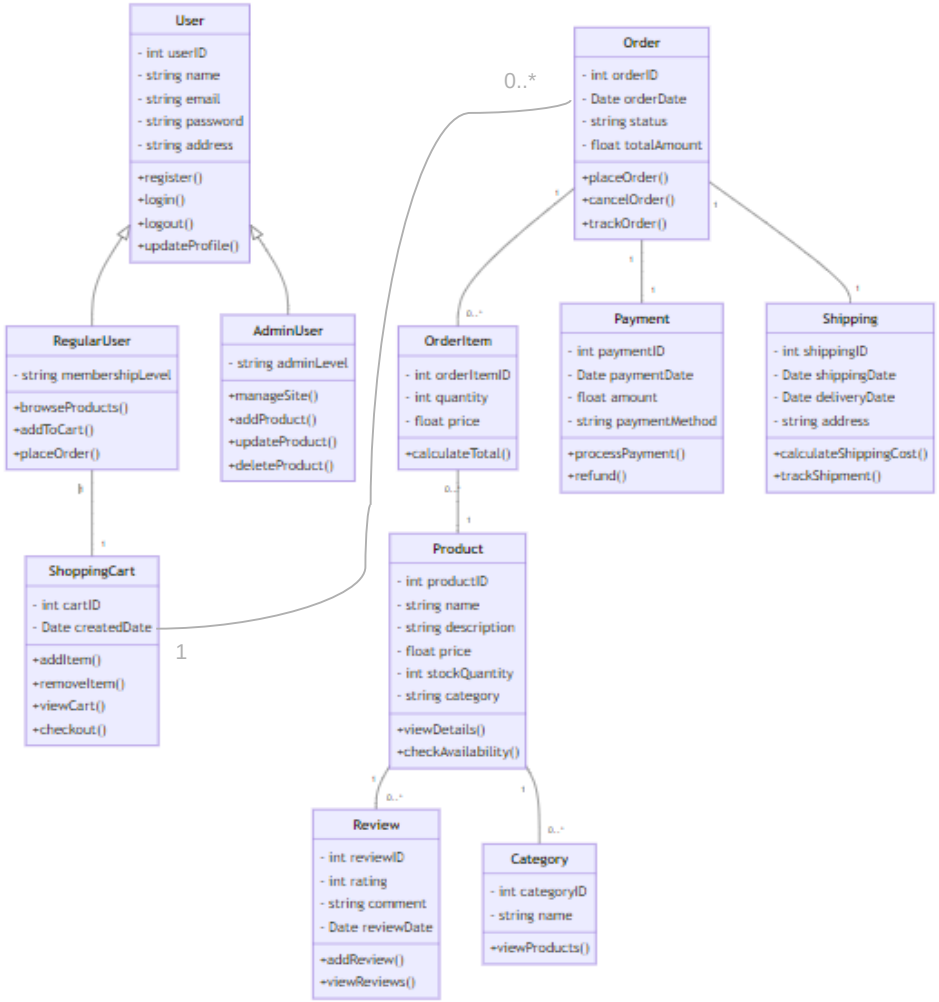
1. **Identify Classes:**
   - Begin by listing all the significant entities in your project.
   - For an online shopping platform, classes might include User, Product, Order, ShoppingCart, Payment, etc.
2. **Define Attributes and Methods:**
   - For each class, list the attributes (data members) and methods (functions) that are relevant.
   - Example:
     - User class:
       - Attributes: userID, name, email, password
       - Methods: register(), login(), logout()
     - Product class:
       - Attributes: productID, name, description, price, stockQuantity
       - Methods: addProduct(), updateProduct(), deleteProduct()

Instructions to draw Class Diagram

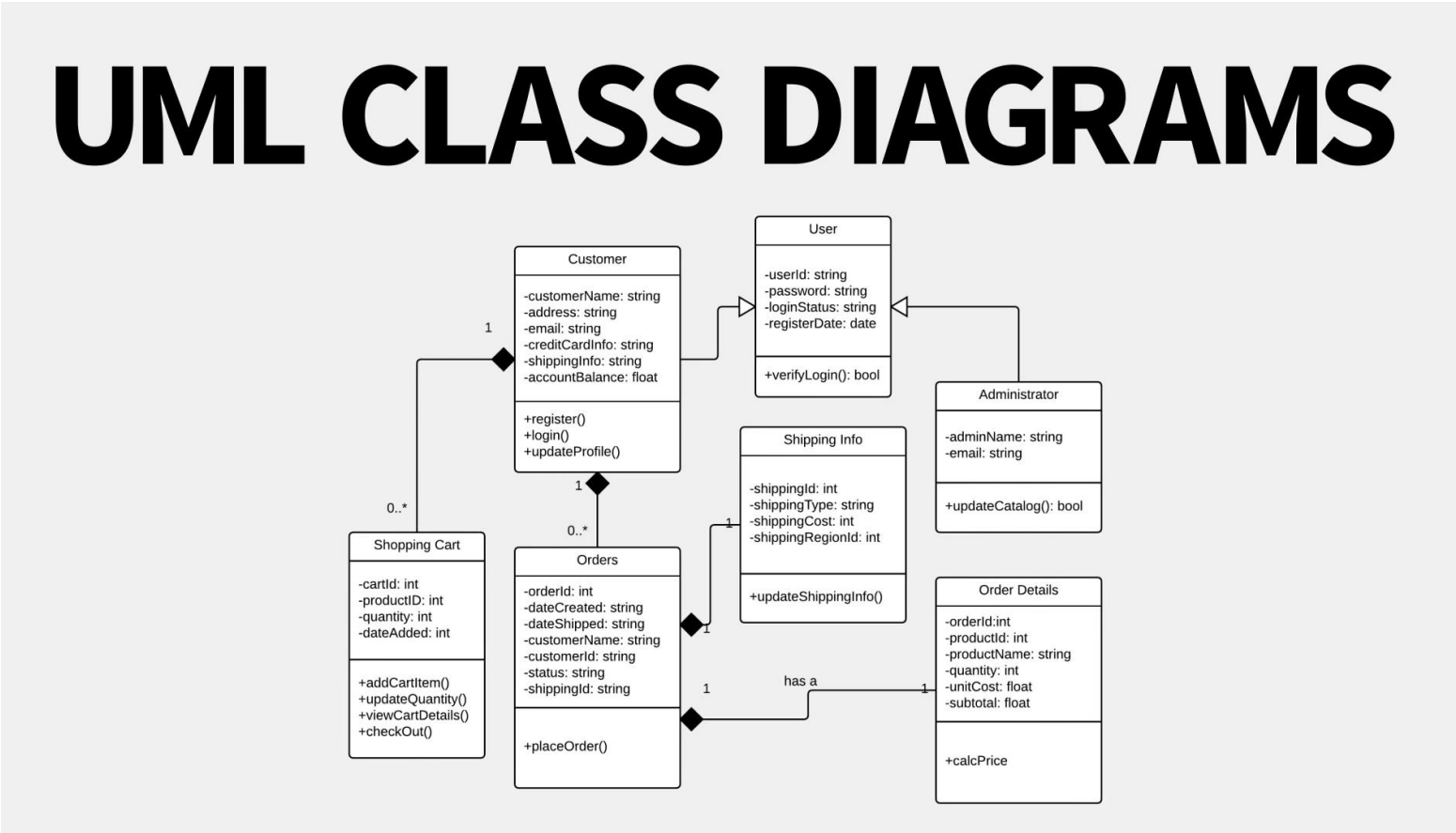
3. **Establish Relationships:**
   - Determine the relationships between classes:
     - **Association:** Represents a relationship between two classes.
       - Example: A User can place an Order.
     - **Aggregation/Composition:** Represents a whole-part relationship.
       - Example: An Order contains multiple Product items.
     - **Inheritance:** Represents a generalization-specialization relationship.
       - Example: AdminUser and RegularUser can inherit from the User class.
4. **Draw the Class Diagram:**
   - Use UML diagram software or drawing tools to create the class diagram.
   - Represent classes as rectangles divided into three sections (name, attributes, methods).
   - Use lines to show relationships:
     - Solid lines for associations with appropriate multiplicity indicators (e.g., 1..*, 0..1).
     - Hollow diamond for aggregation and solid diamond for composition.
     - Arrows for inheritance with the arrow pointing to the base class.

Example

- **ONLINE SHOPPING SYSTEM**


User
- int userID
- string name
- string email
- string password
- string address
+register()
+login()
+logout()
+updateProfile()
Order
- int orderID
- Date orderDate
- string status
- float totalAmount
+placeOrder()
+cancelOrder()
+trackOrder()
0..*
RegularUser
- string membershipLevel
+browseProducts()
+addToCart()
+placeOrder()
AdminUser
- string adminLevel
+manageSite()
+addProduct()
+updateProduct()
+deleteProduct()
OrderItem
- int orderItemID
- int quantity
- float price
+calculateTotal()
Payment
- int paymentID
- Date paymentDate
- float amount
- string paymentMethod
+processPayment()
+refund()
Shipping
- int shippingID
- Date shippingDate
- Date deliveryDate
- string address
+calculateShippingCost()
+trackShipment()
ShoppingCart
- int cartID
- Date createdDate
+addItem()
+removeItem()
+viewCart()
+checkout()
1
Product
- int productID
- string name
- string description
- float price
- int stockQuantity
- string category
+viewDetails()
+checkAvailability()
Review
- int reviewID
- int rating
- string comment
- Date reviewDate
+addReview()
+viewReviews()
Category
- int categoryID
- string name
+viewProducts()

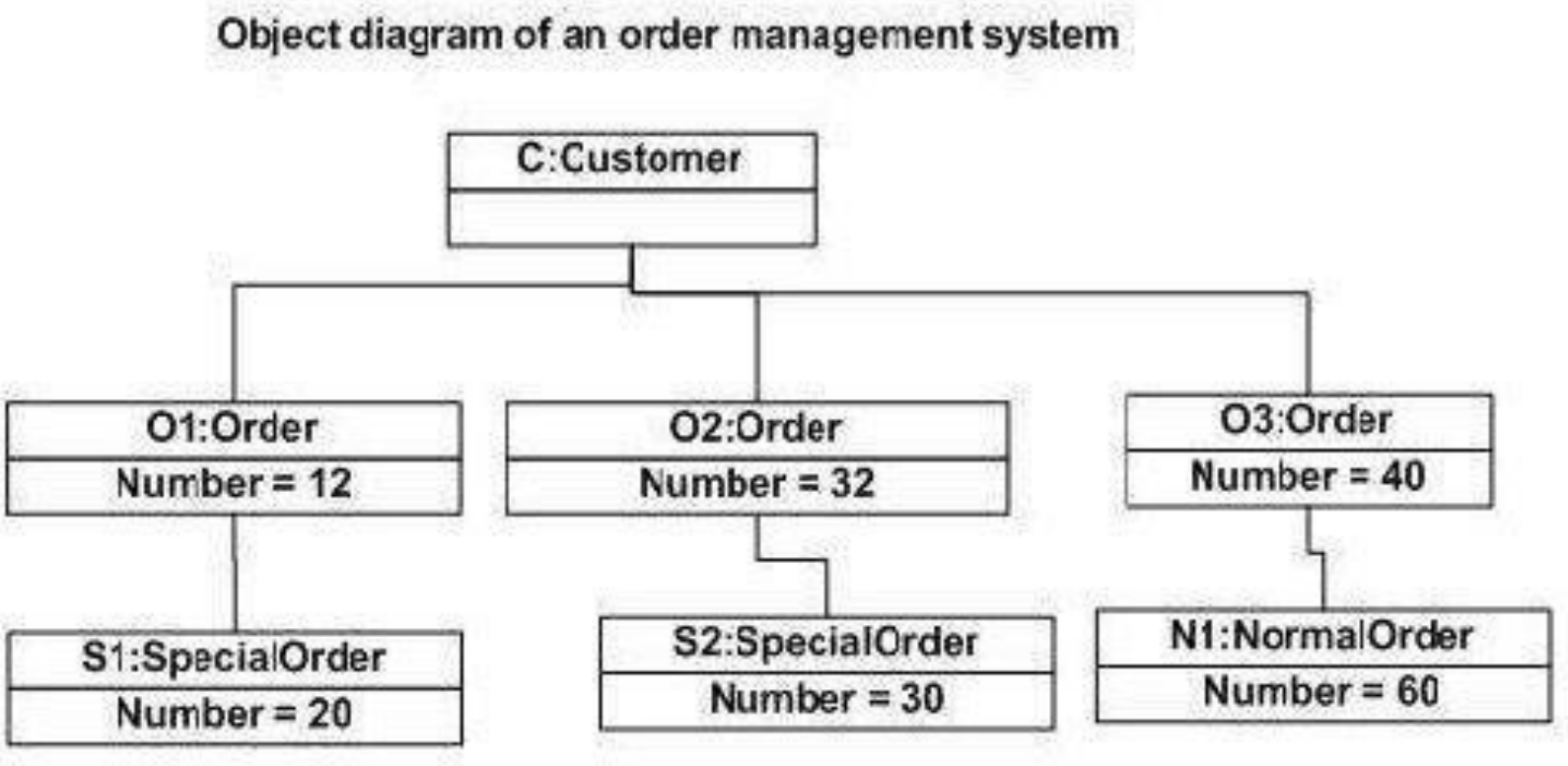
Class diagram – Food Ordering application



Usage of Class diagrams

- To describe the static view of a system.
- To show the collaboration among every instance in the static view.
- To describe the functionalities performed by the system.
- To construct the software application using object-oriented languages.

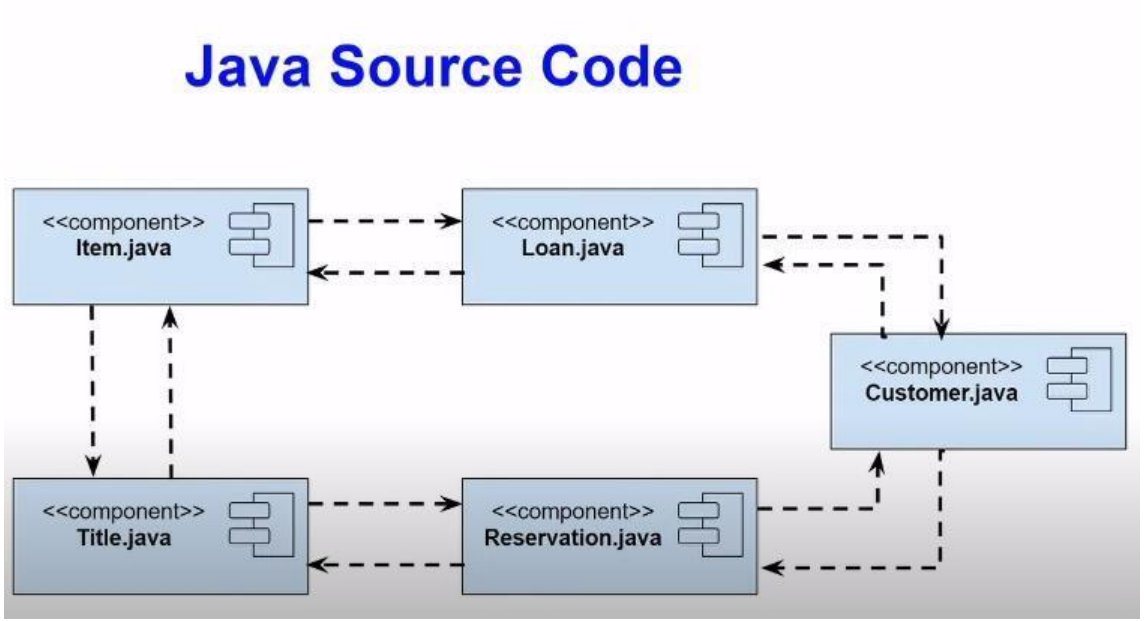
Object Diagrams - instance of a class diagram



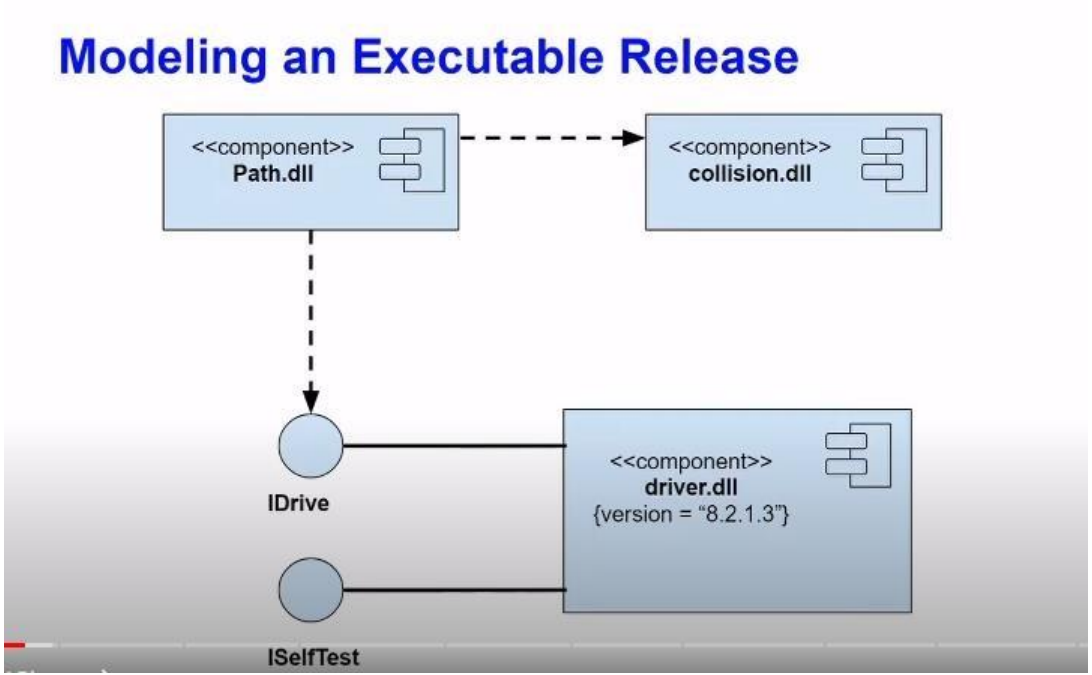
Component Diagram

- It models the physical view of a system such as executables, files, libraries, etc. that resides within the node.
- Following are some artifacts that are needed to be identified before drawing a component diagram:
- What files are used inside the system?
- What is the application of relevant libraries and artifacts?
- What is the relationship between the artifacts?
- The component diagram can be used for the followings:
- To model the components of the system.
- To model the schemas of a database.
- To model the applications of an application.
- To model the system's source code.

Java Source Code

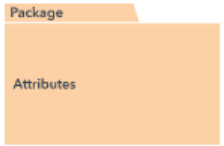


Modeling an Executable Release



Component Diagrams - Notations

Component symbol



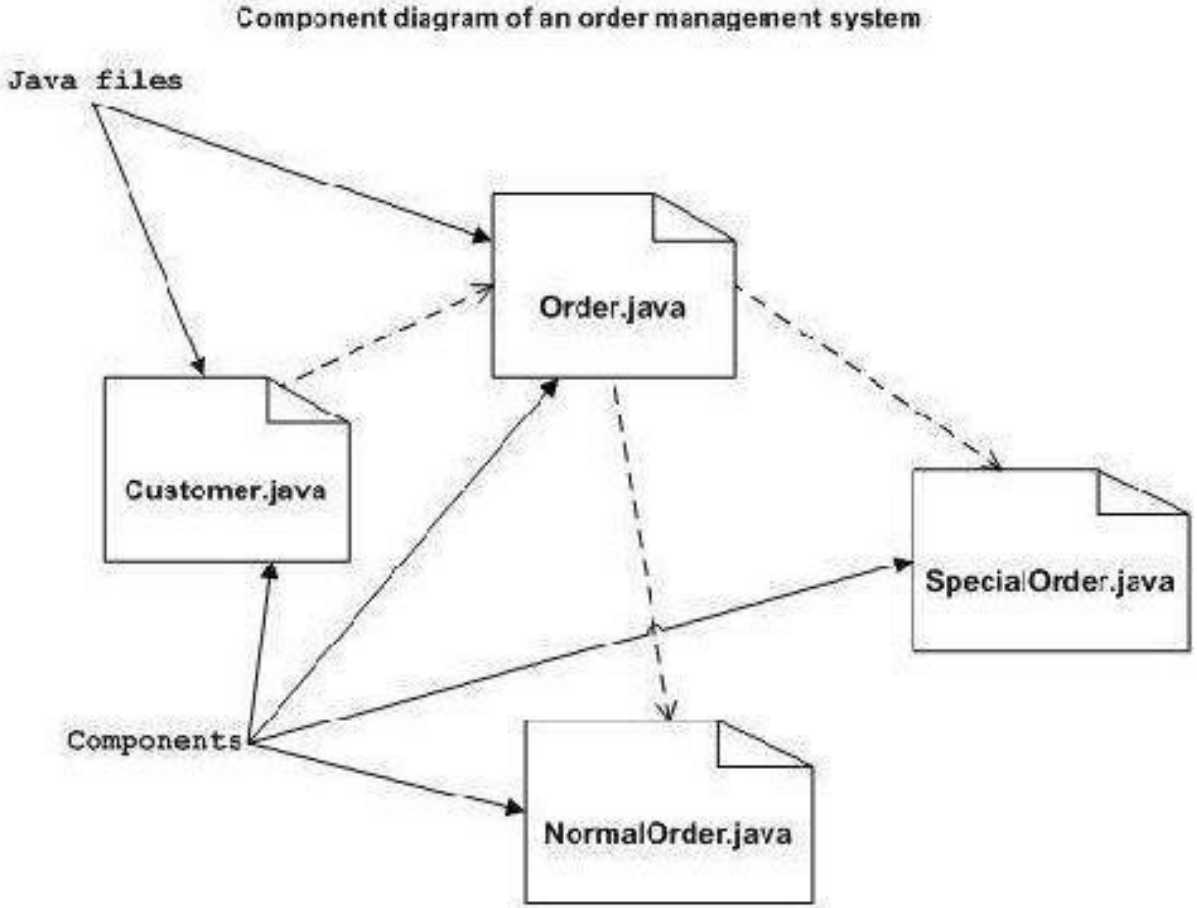
Package symbol

Note symbol

Node symbol

Dependency symbol

Component Diagrams - model the physical aspects of a system



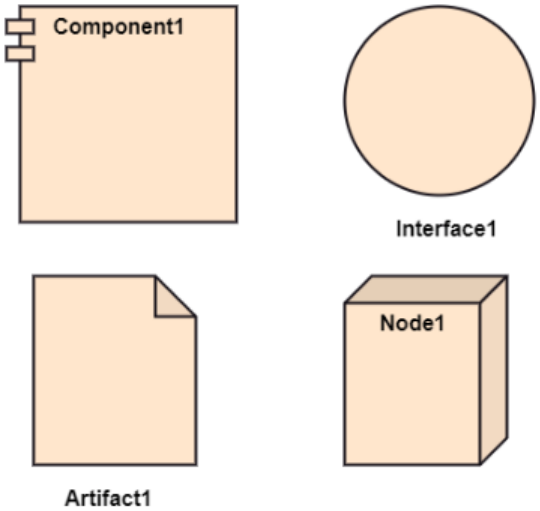
Deployment Diagrams

- To visualize the topology of the physical components of a system, where the software components are deployed

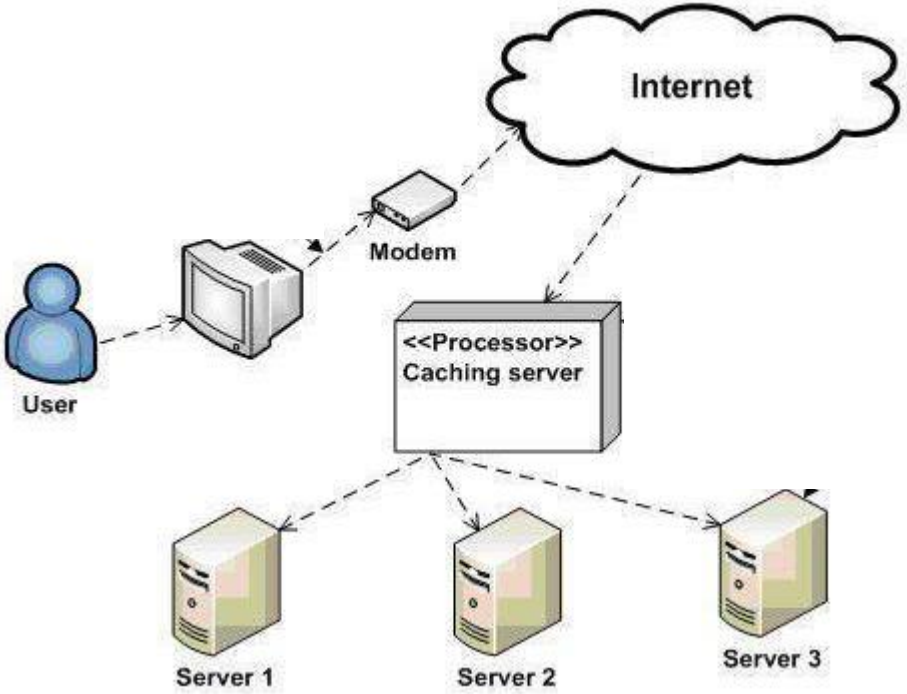
Symbol and notation of Deployment diagram

The deployment diagram consist of the following notations:

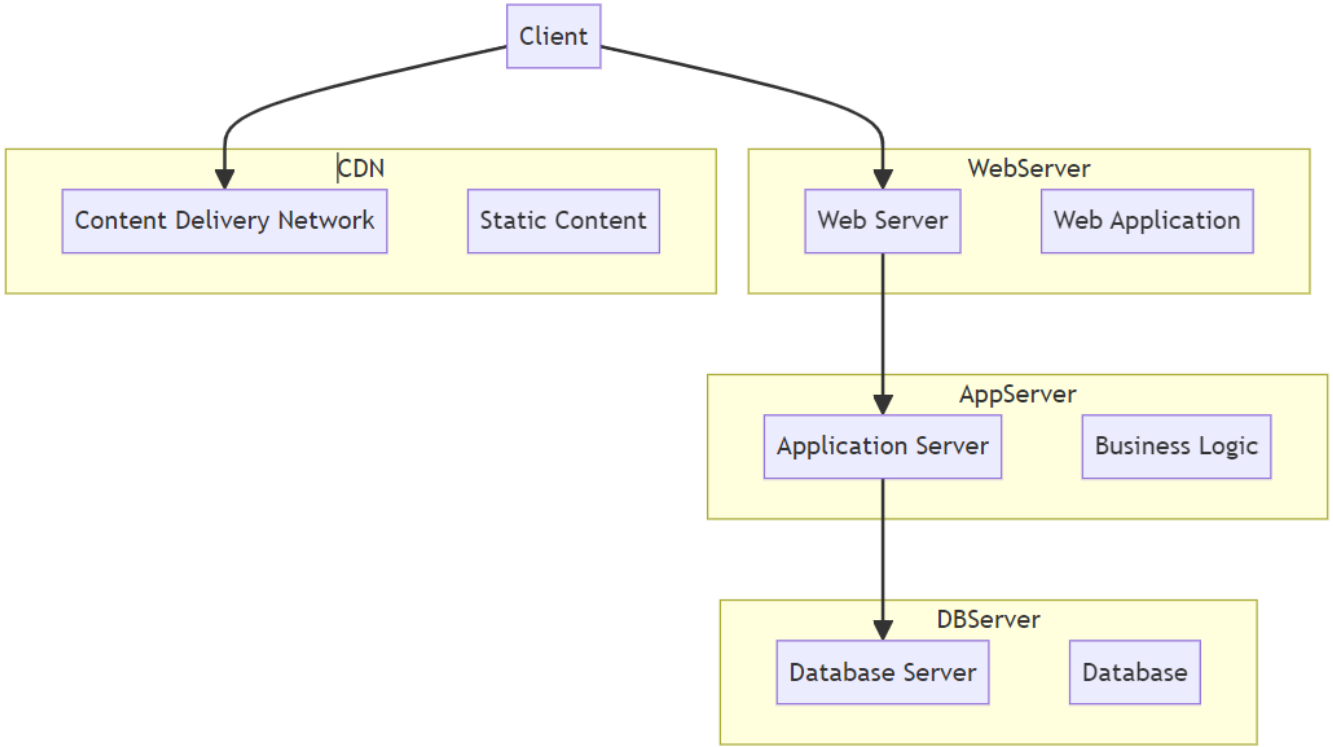
1. A component
2. An artifact
3. An interface
4. A node



Deployment Diagrams



Deployment Diagrams of a Web Application



When to use a Deployment Diagram?

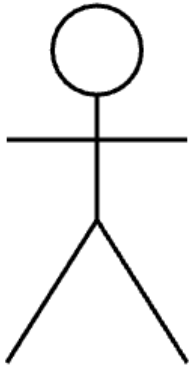
- To model the network and hardware topology of a system.
- To model the distributed networks and systems.
- Implement forwarding and reverse engineering processes.
- To model the hardware details for a client/server system.
- For modeling the embedded system.

Behavioral Diagrams - Behaviour of the system when it is running/operating

- Use case diagram
- Sequence diagram
- Collaboration diagram
- State chart diagram
- Activity diagram

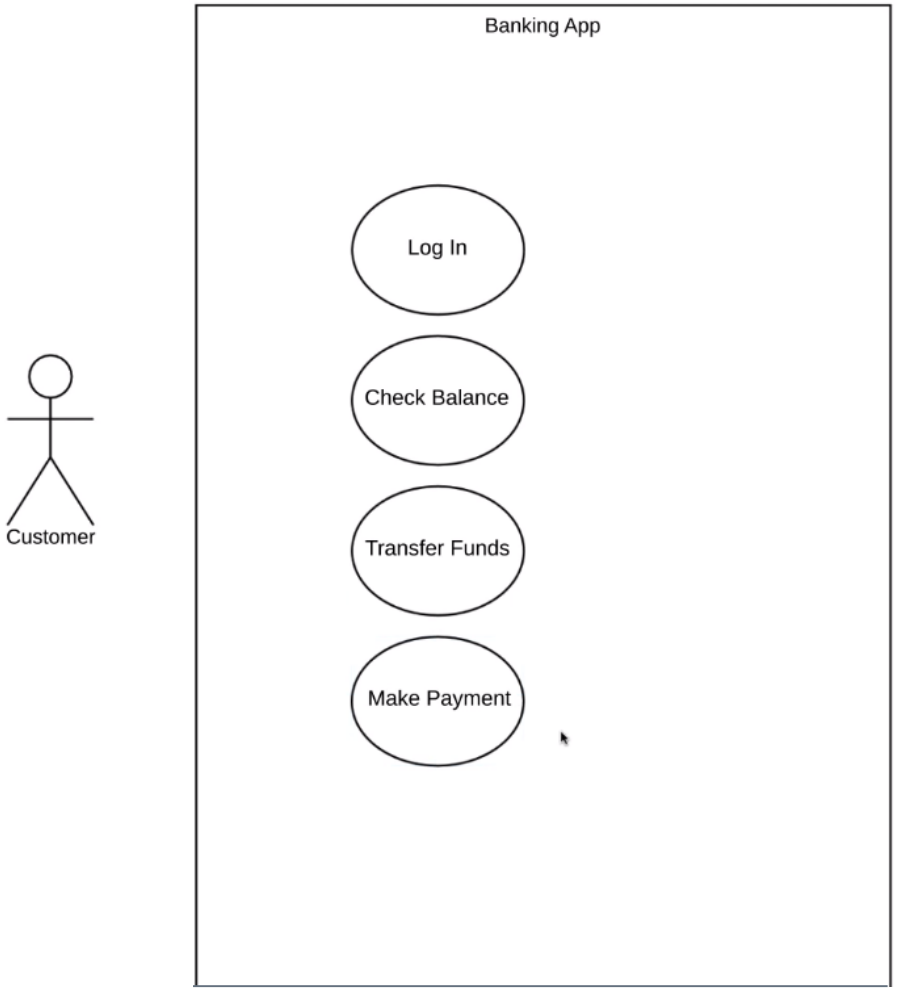
Use Case Actors

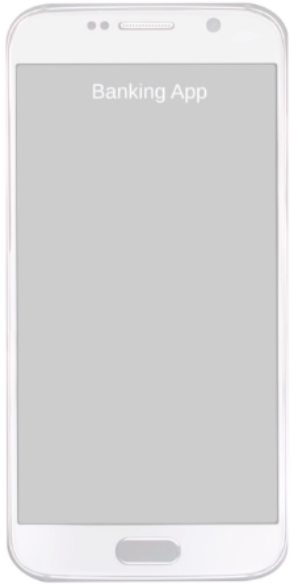
- Primary Actors
  - Initiate the use of the system



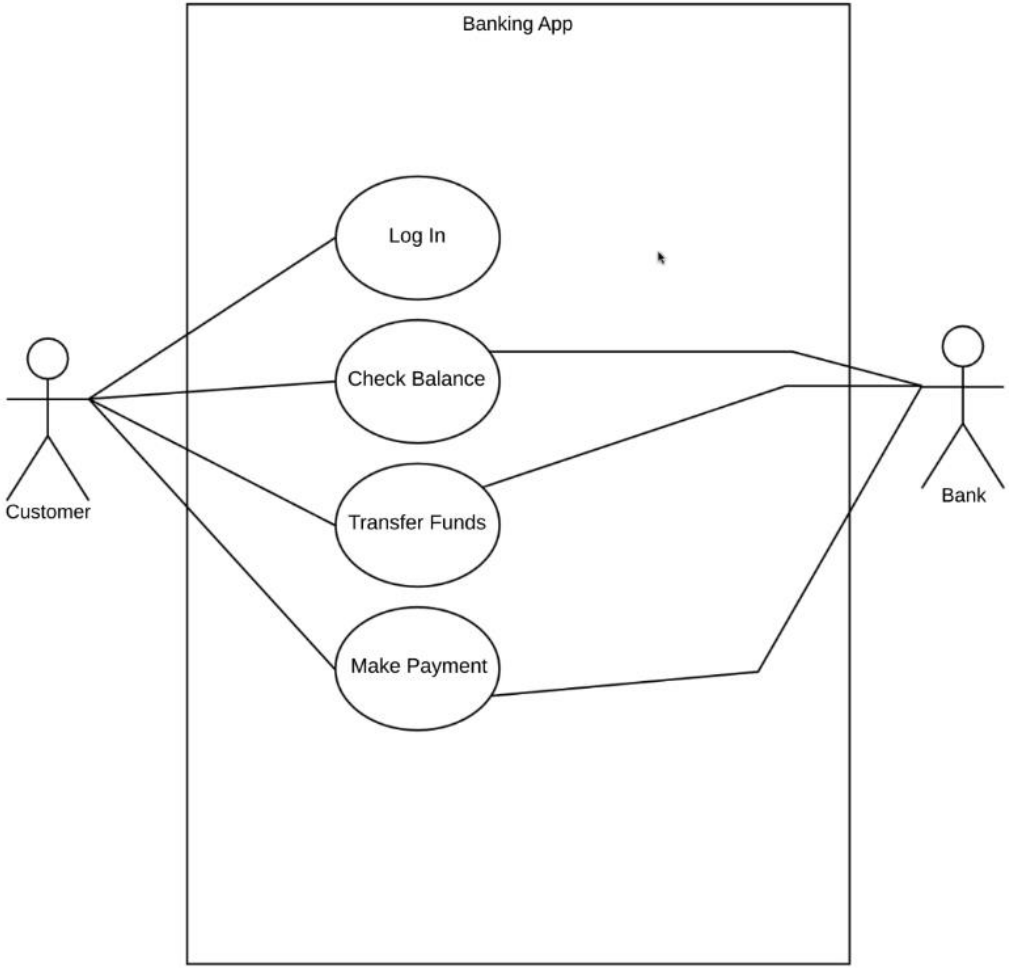
- Secondary Actors
  - Reactionary

- Actor can be
  - People
  - Organization
  - External Device





Log In
Check Balance
Transfer Funds
Make Payment



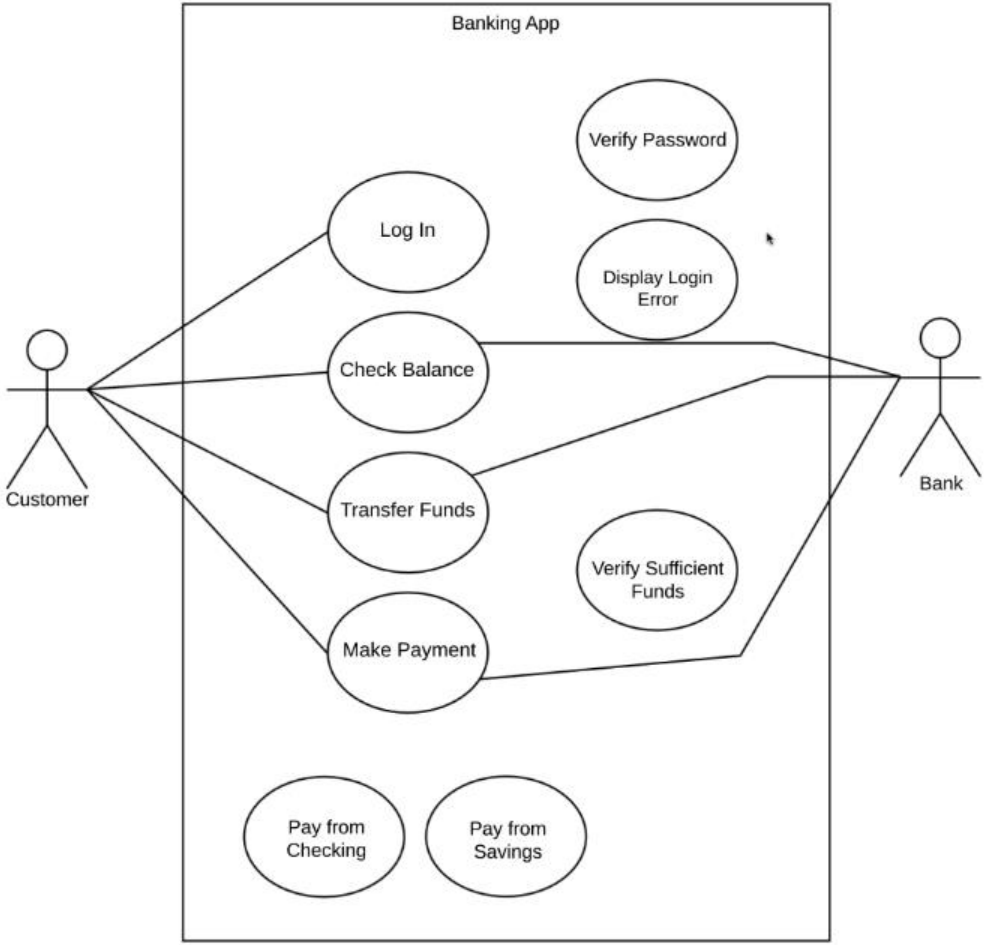
Relationships

Association

Include

Extend

Generalization



Relationships

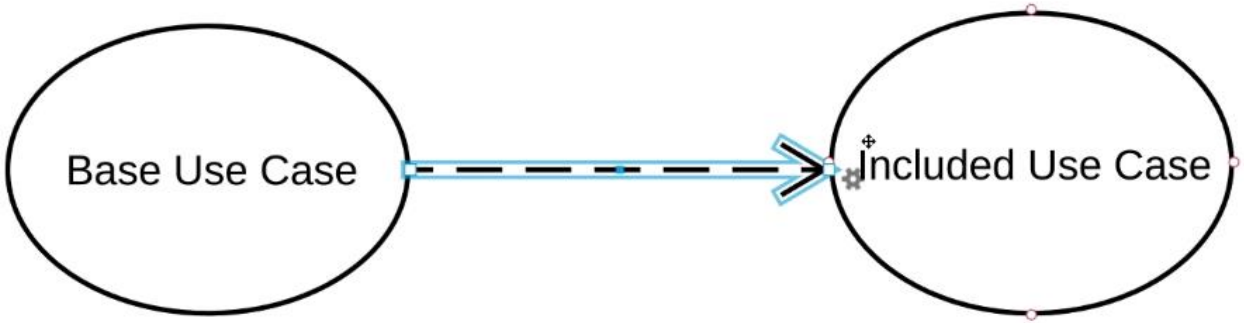
Association

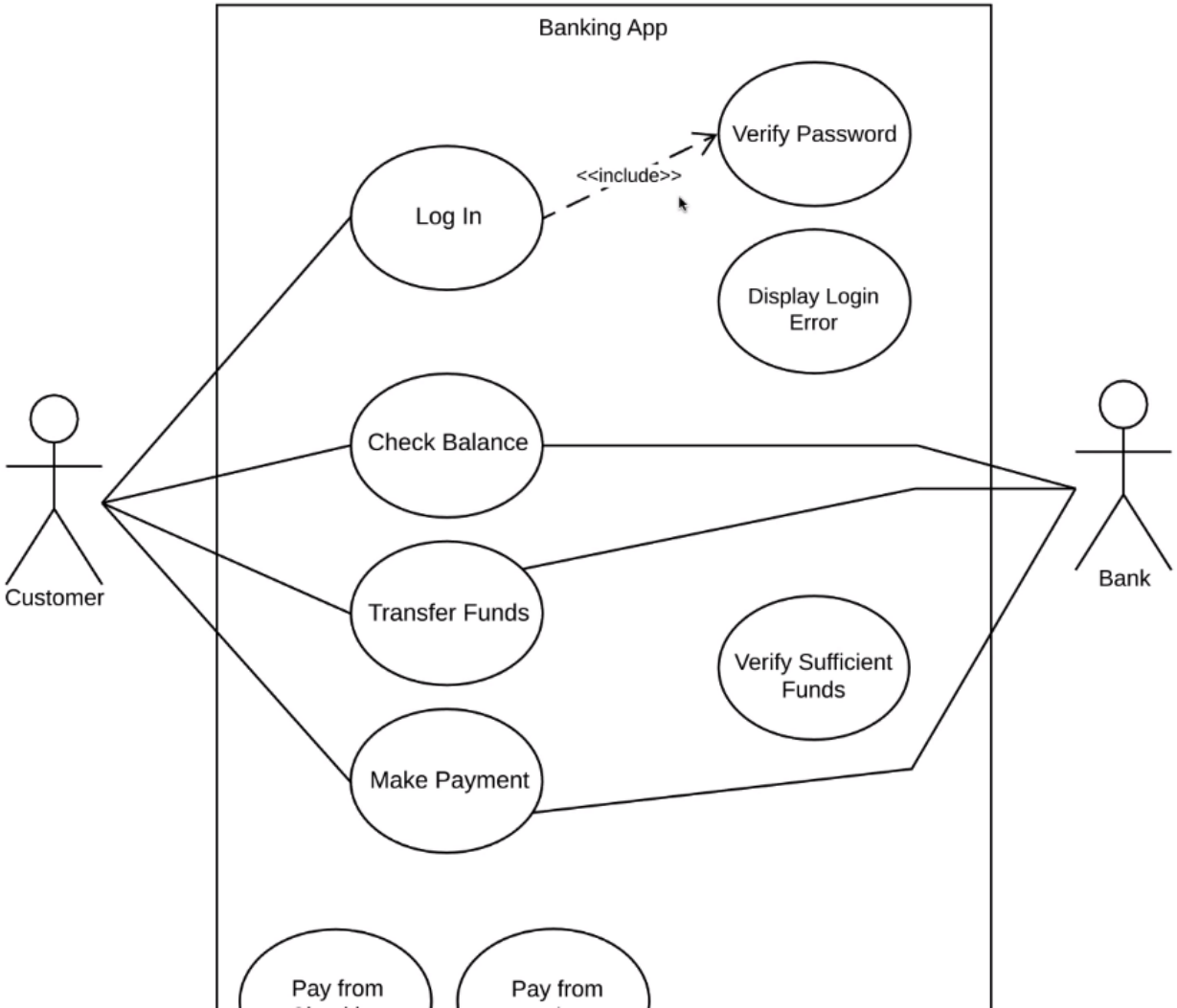
Include

Extend

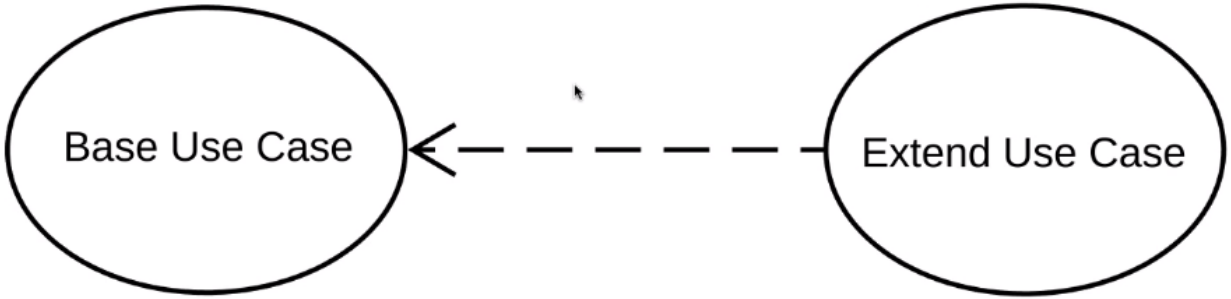
Generalization

Include

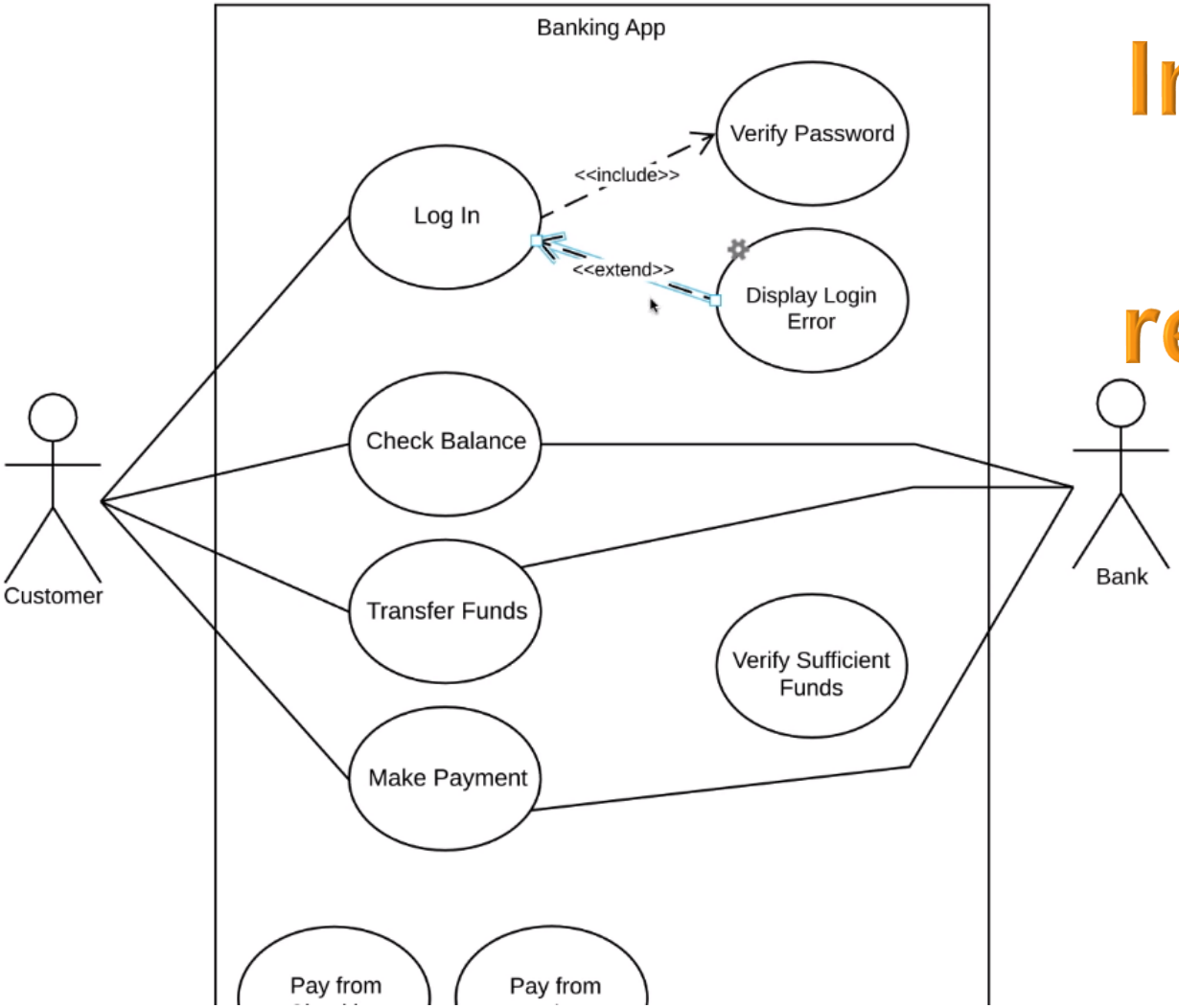


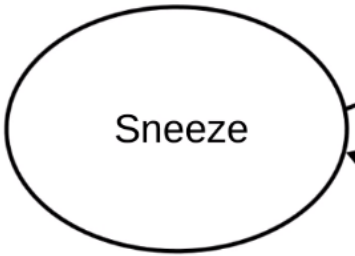

Banking App
Verify Password
<<include>>
Log In
Display Login Error
Check Balance
Customer
Bank
Transfer Funds
Verify Sufficient Funds
Make Payment
Pay from
Pay from

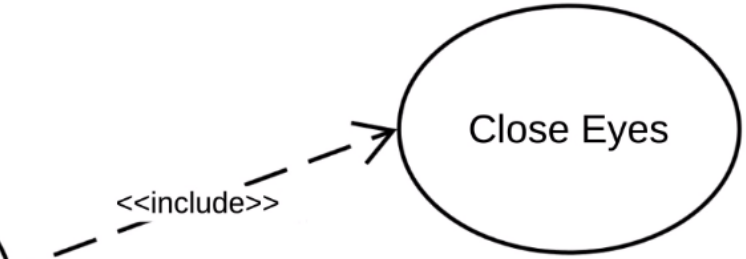
Extend

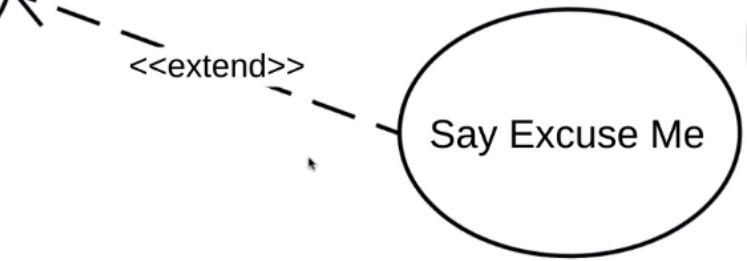


Include and extend relationship



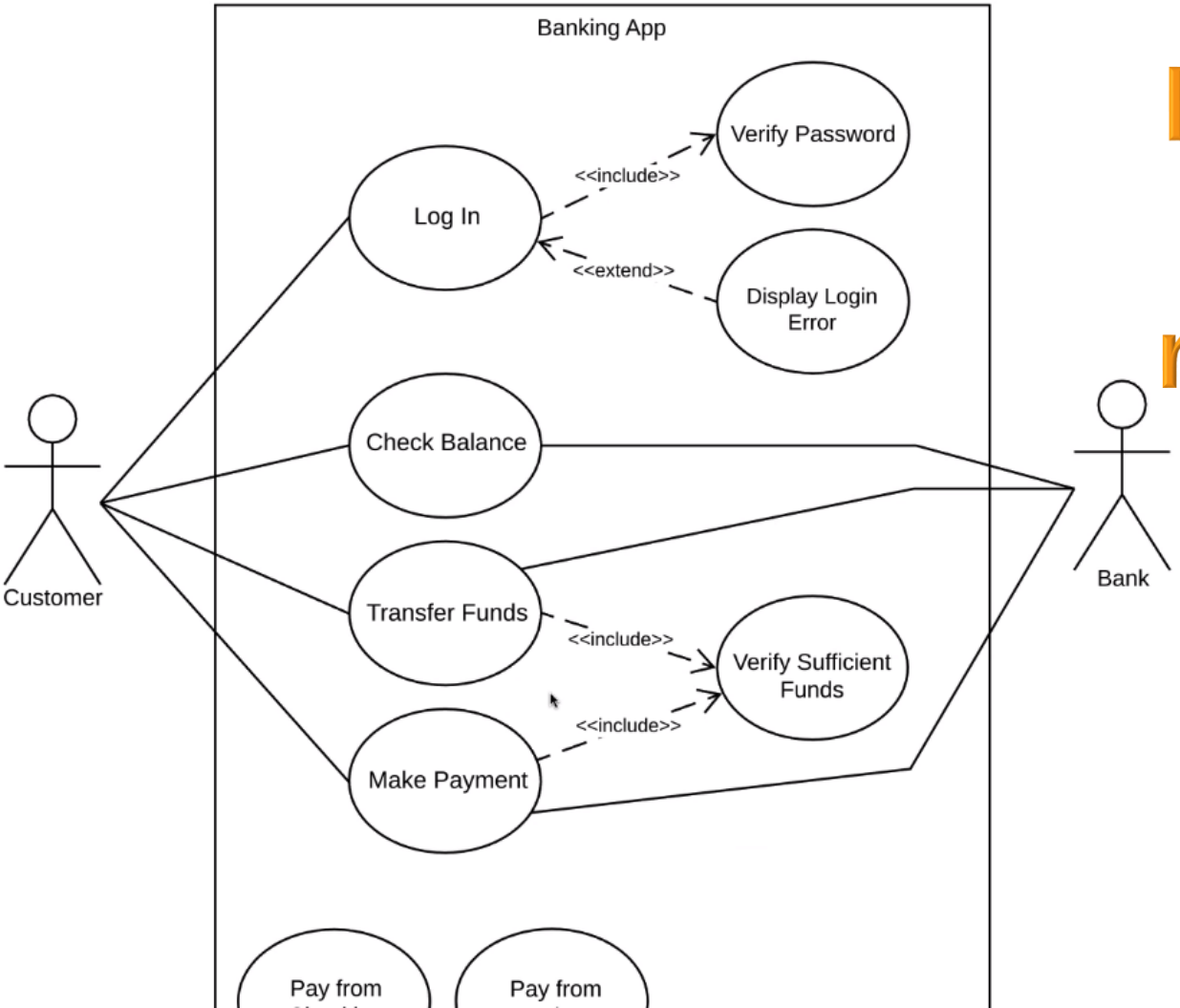

Sneeze


<<include>>

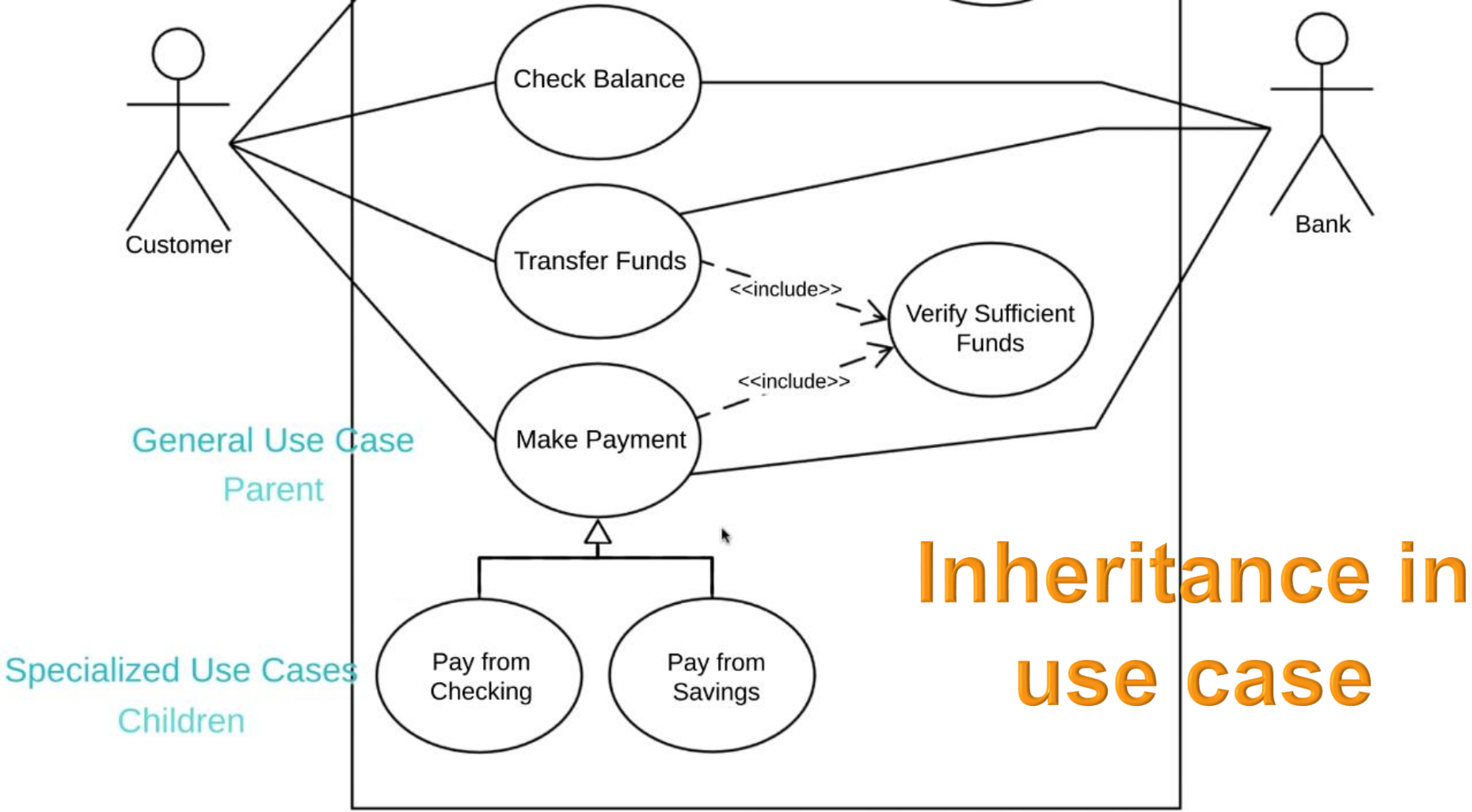

<<extend>>

Close Eyes

Say Excuse Me

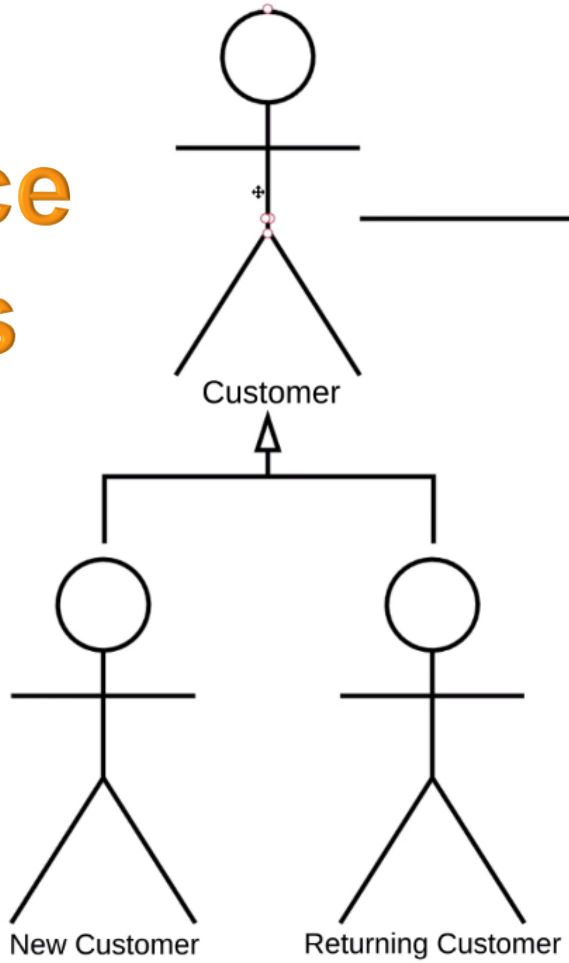


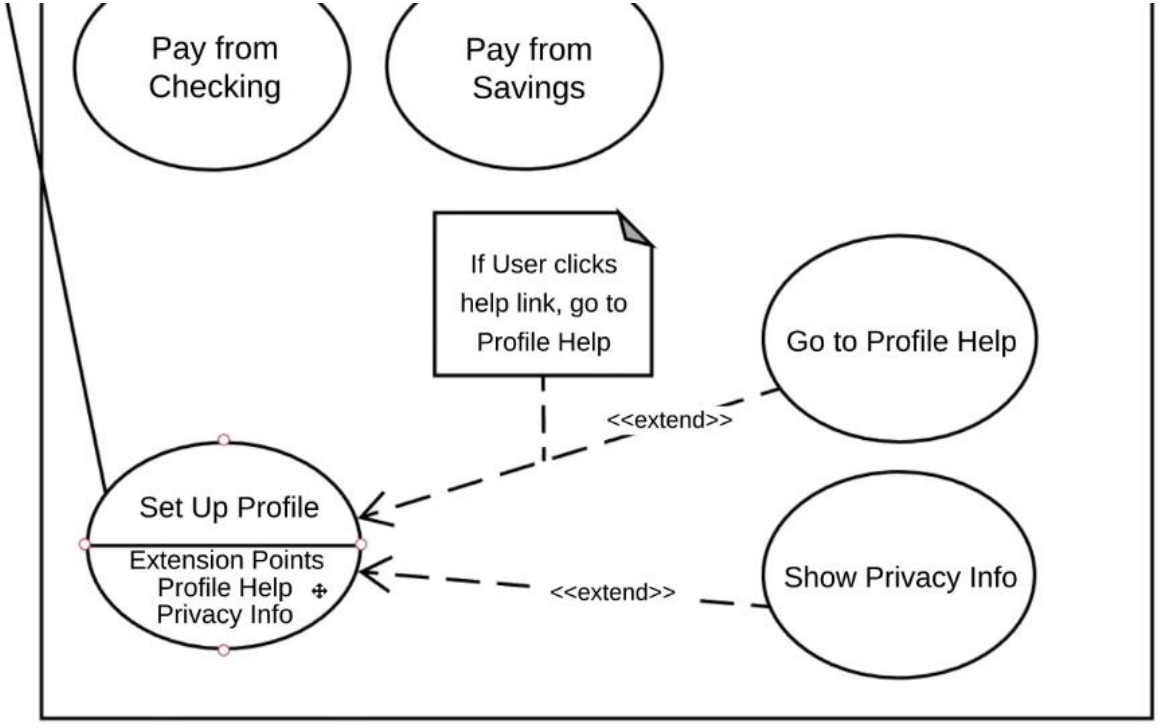
Include and extend relationship



Inheritance in use case

Inheritance in Actors





Extension Point

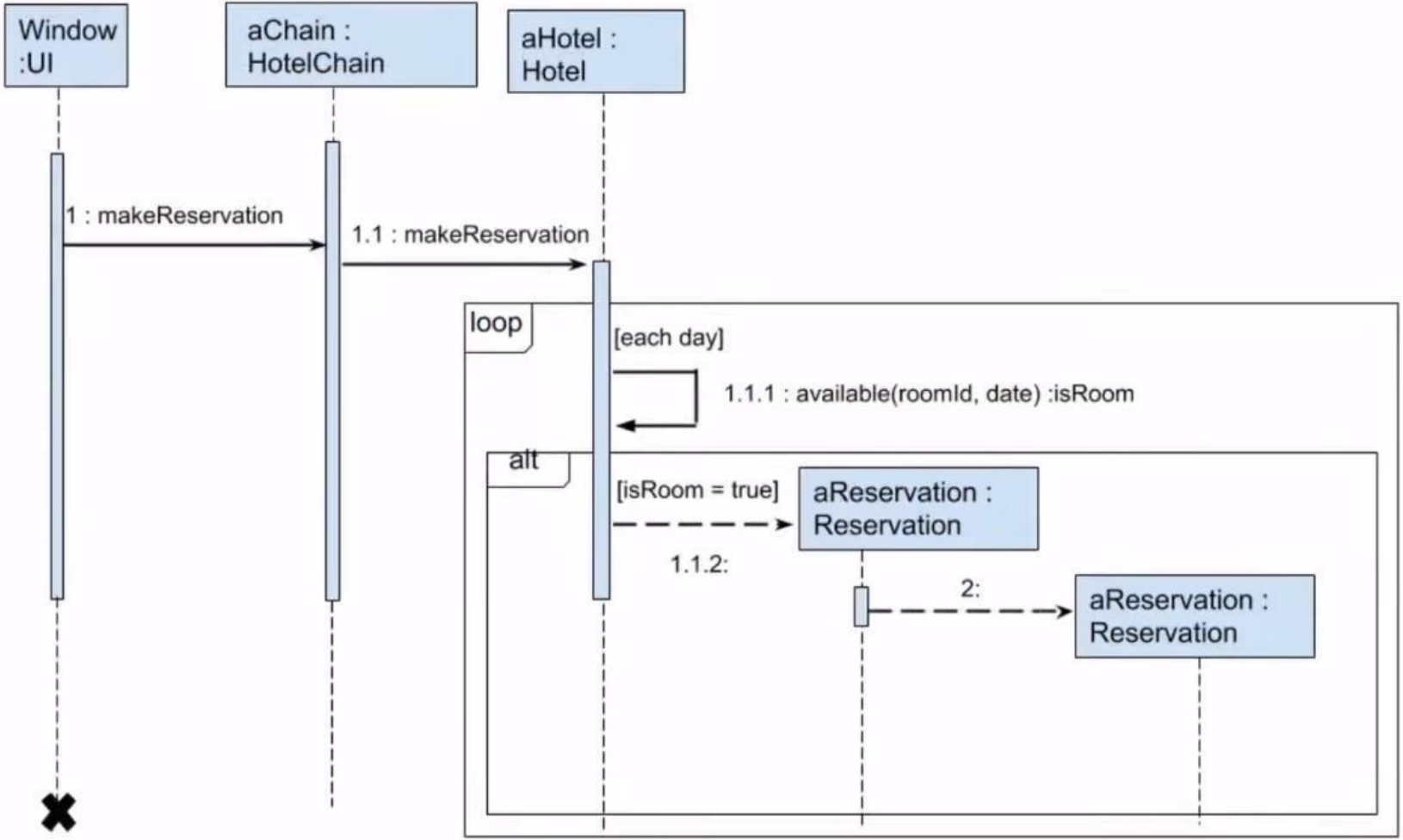
Interaction Diagrams - Interactions among the different elements in the model

Sequence diagram

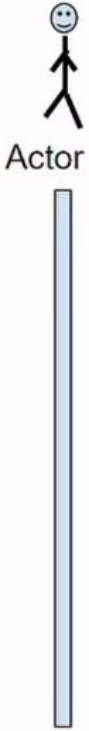
Collaboration diagram

Purpose of a Sequence Diagram

- To model high-level interaction among active objects within a system.
- To model interaction among objects inside a collaboration realizing a use case.
- It either models generic interactions or some certain instances of interaction.

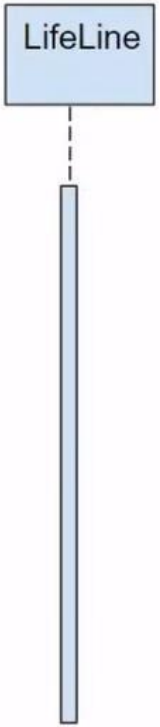

Window
:UI
aChain :
HotelChain
aHotel :
Hotel
1 : makeReservation
1.1 : makeReservation
loop
[each day]
1.1.1 : available(roomId, date) :isRoom
alt
[isRoom = true]
aReservation :
Reservation
1.1.2:
2:
aReservation :
Reservation

Actor



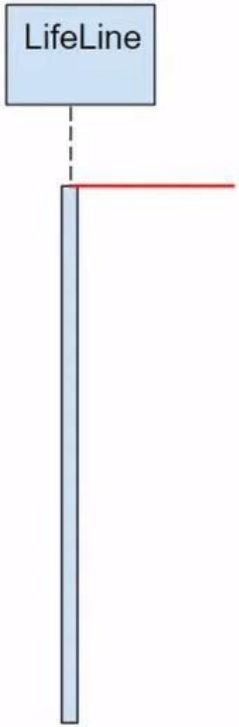
- A type of role played by the instance that interacts with the subject (for example, by exchanging signals and data).
- Actor represents the roles played by users, external equipment, or other actors.
- Actor does not necessarily represent a specific physical entity, but simply a specific role of some entity.
- A person can play the role of several different actors, and, conversely, an actor may be played by multiple different people.

LifeLine



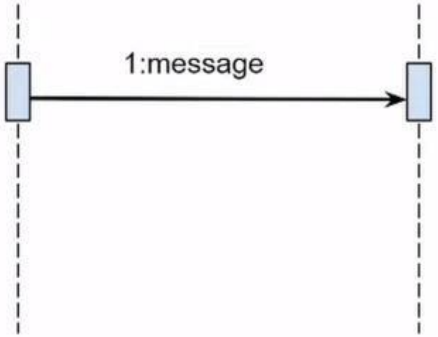
Represents an individual participant in the interaction and displays the passage of time.

Activation



- The thin rectangle on the lifeline indicates the period during which the element performs an operation.

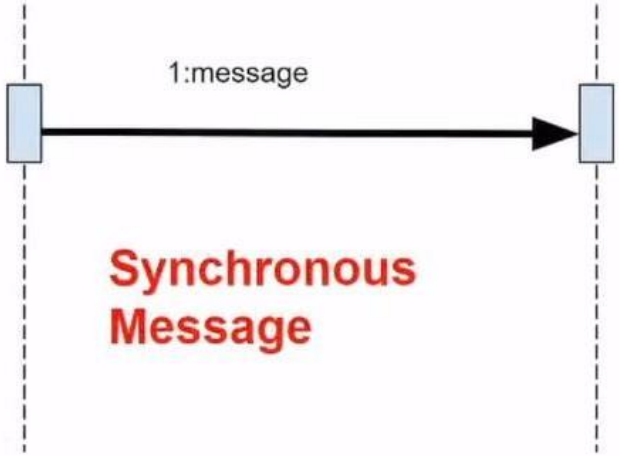
- Top and bottom of the rectangle are aligned with the start and end of a specific function.



Call Message

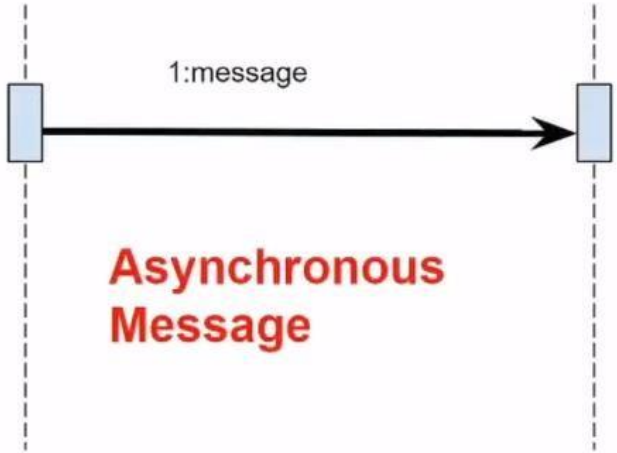
- Used to call procedures, execute operations, or designate individual nested control flows.
- One end of the arrow always touches the control focus or lifeline of the client object that triggers the message.
- The tip of the arrow points at the lifeline of the object that receives the message and takes action as a result. At the same time, this object also often receives a control focus, becoming active.

Call Message



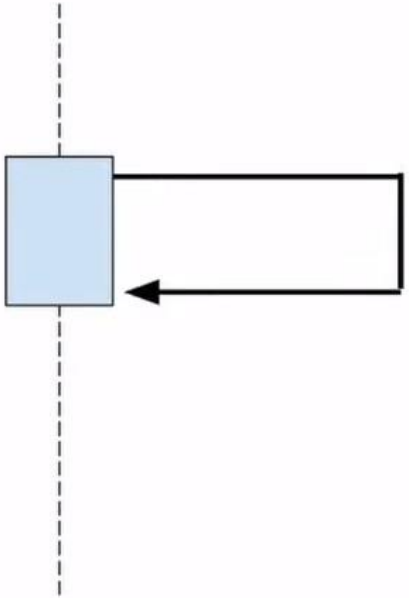
The message is synchronous (depicted with a filled arrow), when the client sends a message, for example, to the server and waits for a response before doing anything else.

Call Message



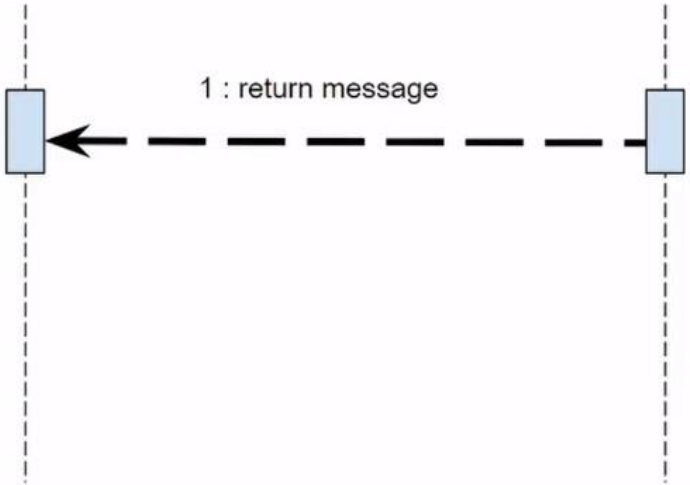
Message is asynchronous (depicted with a regular arrow), when the client continues to perform operations without waiting for a response.

Self Message



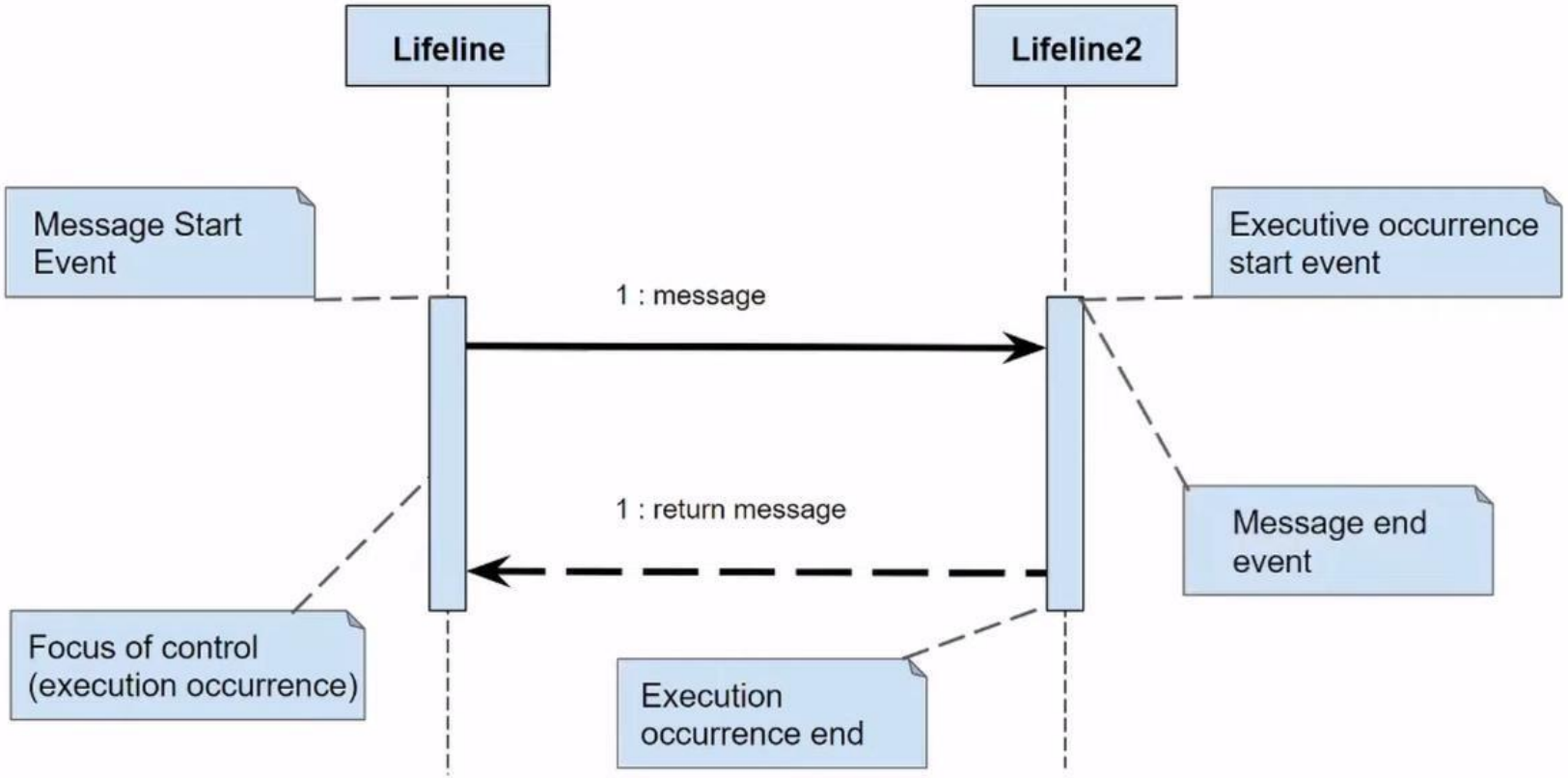
A participant sends a message (command) to itself.

Return Message

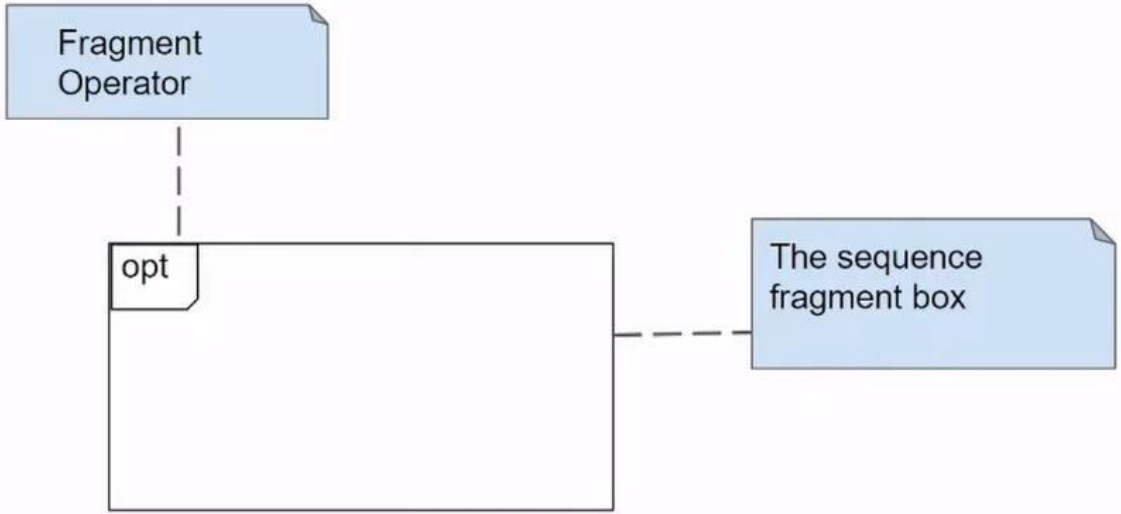


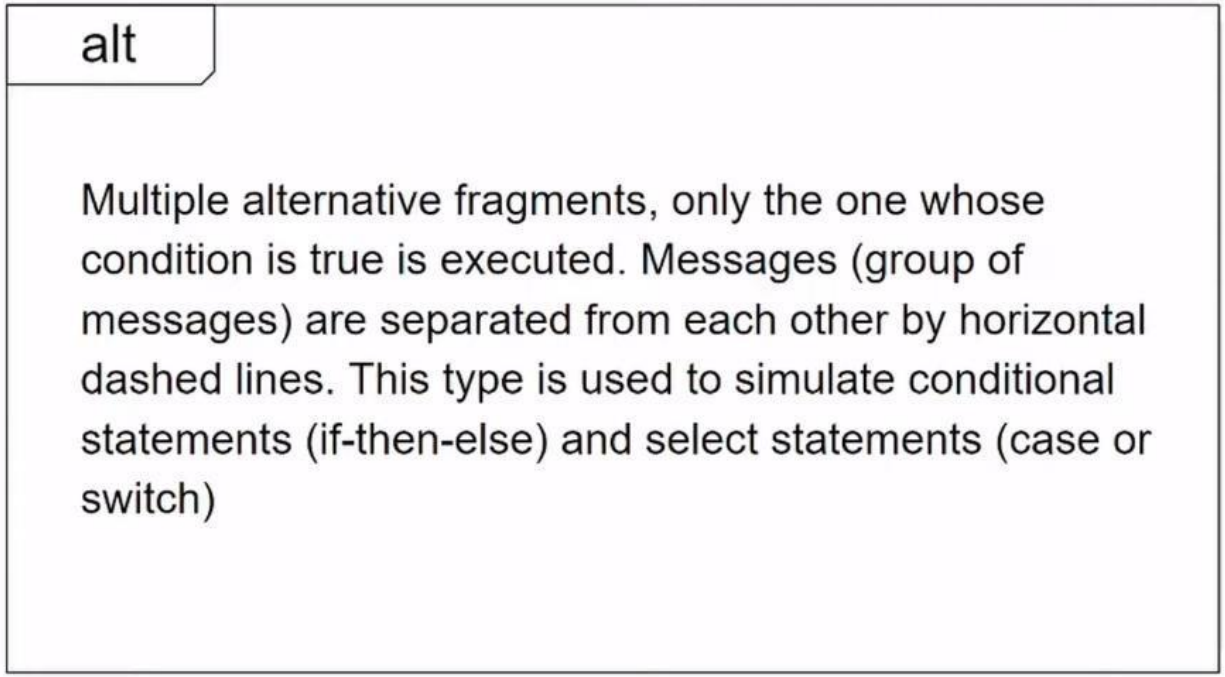
Return message identifies the specific communication between the lifelines of an interaction and represents the transfer of information to the caller of the corresponding message.

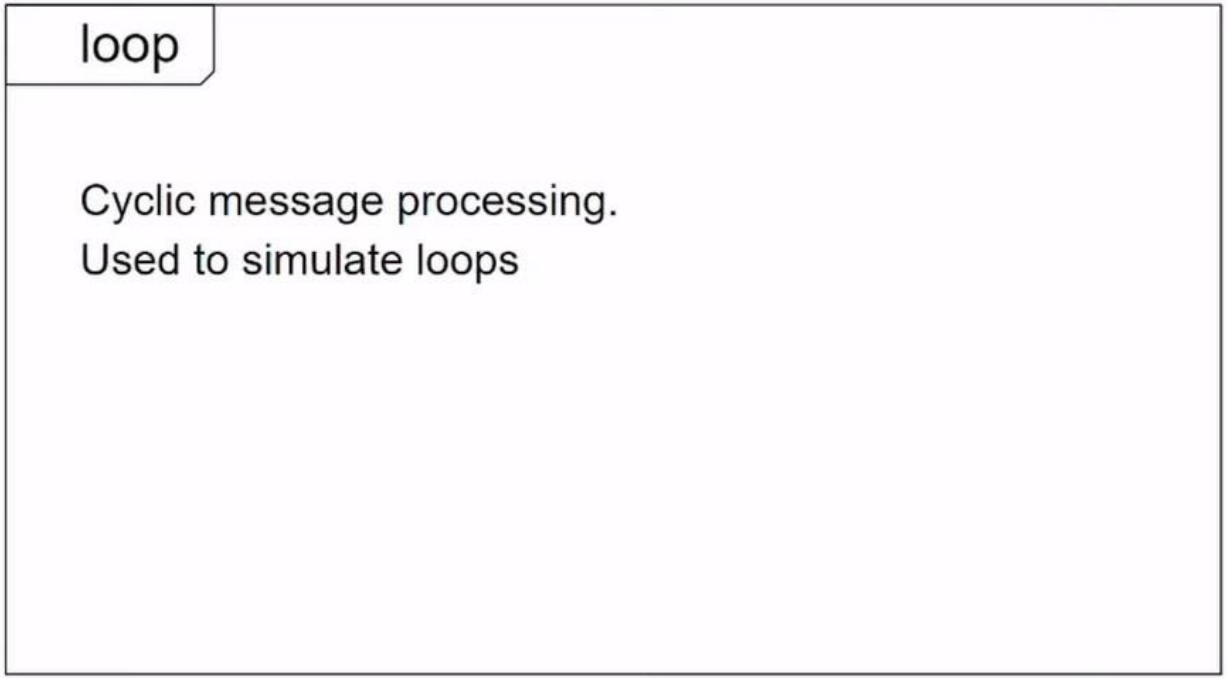
Focus of Control

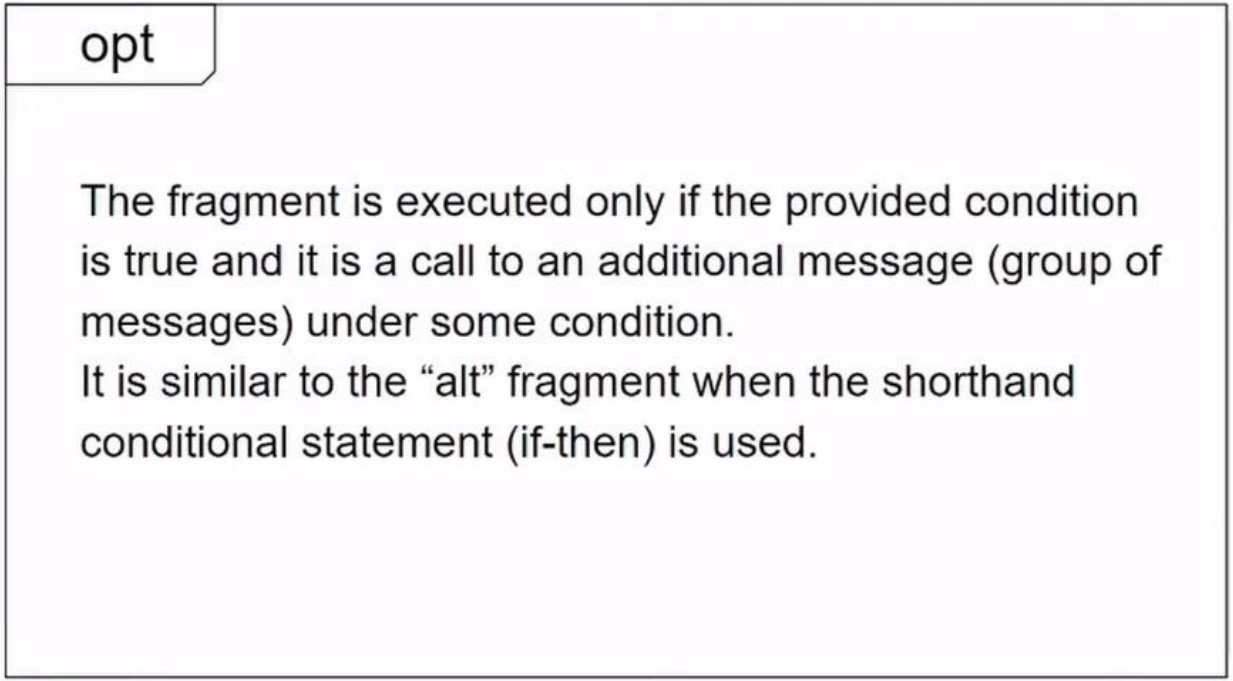


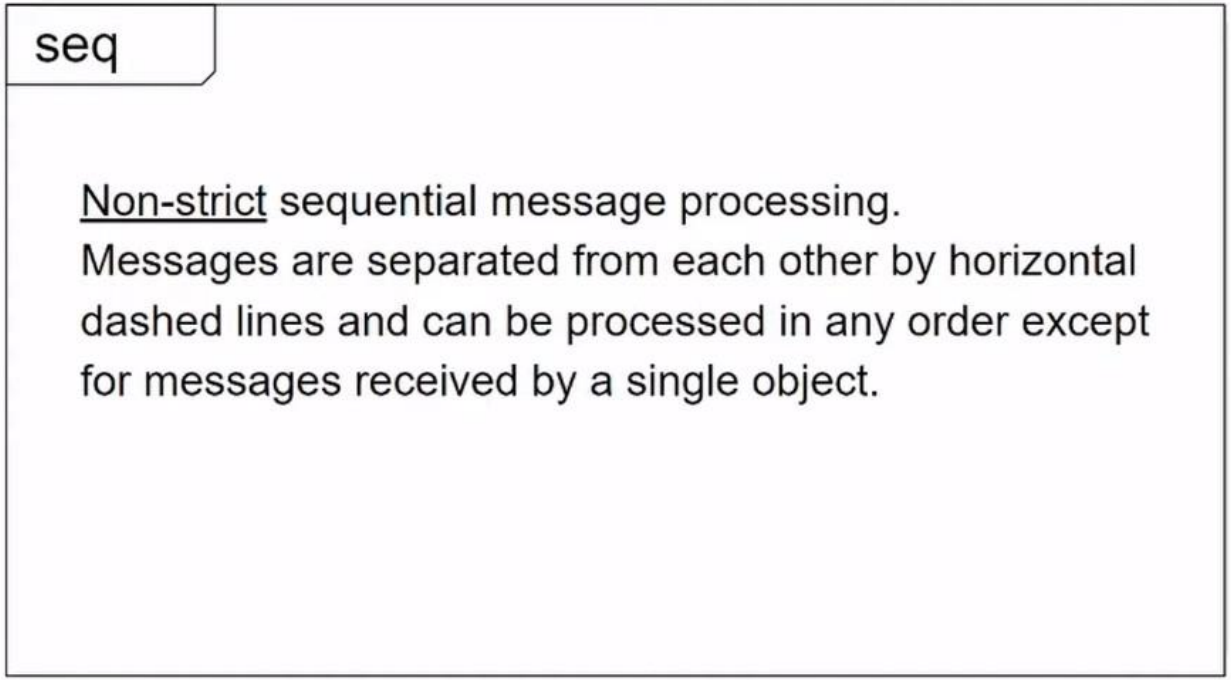
Sequence Fragments

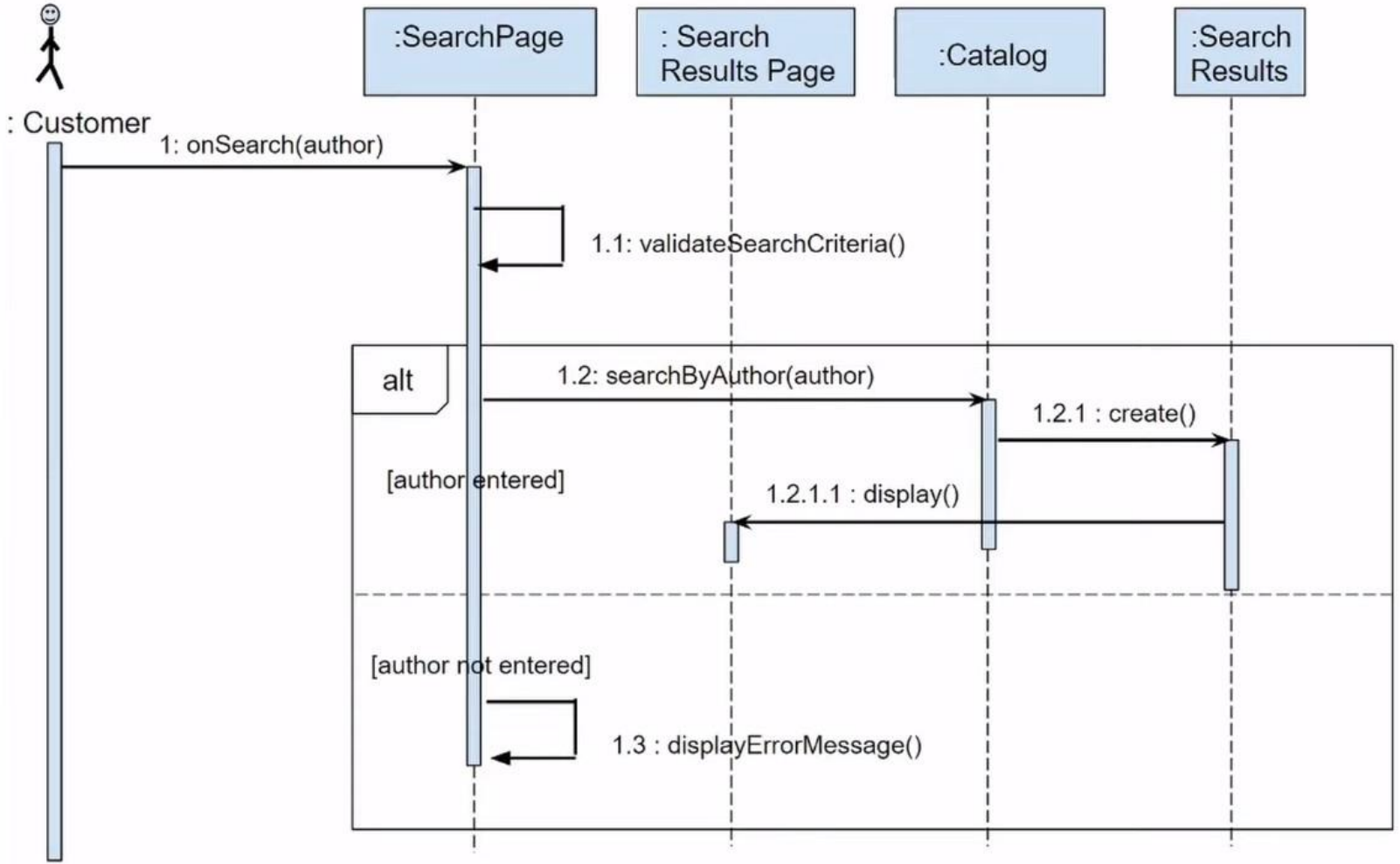



alt
Multiple alternative fragments, only the one whose condition is true is executed. Messages (group of messages) are separated from each other by horizontal dashed lines. This type is used to simulate conditional statements (if-then-else) and select statements (case or switch)


loop
Cyclic message processing.
Used to simulate loops

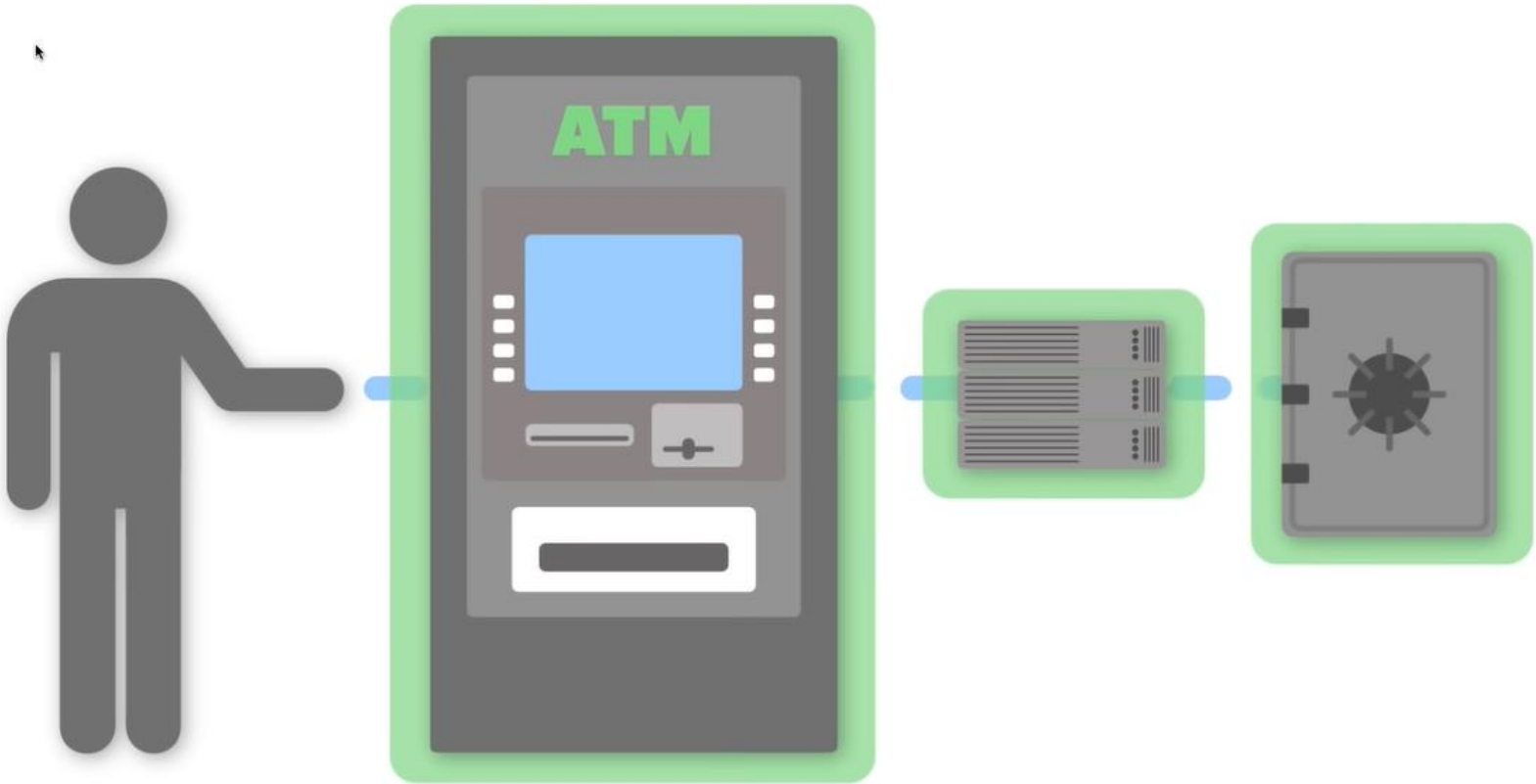

opt
The fragment is executed only if the provided condition is true and it is a call to an additional message (group of messages) under some condition.
It is similar to the “alt” fragment when the shorthand conditional statement (if-then) is used.

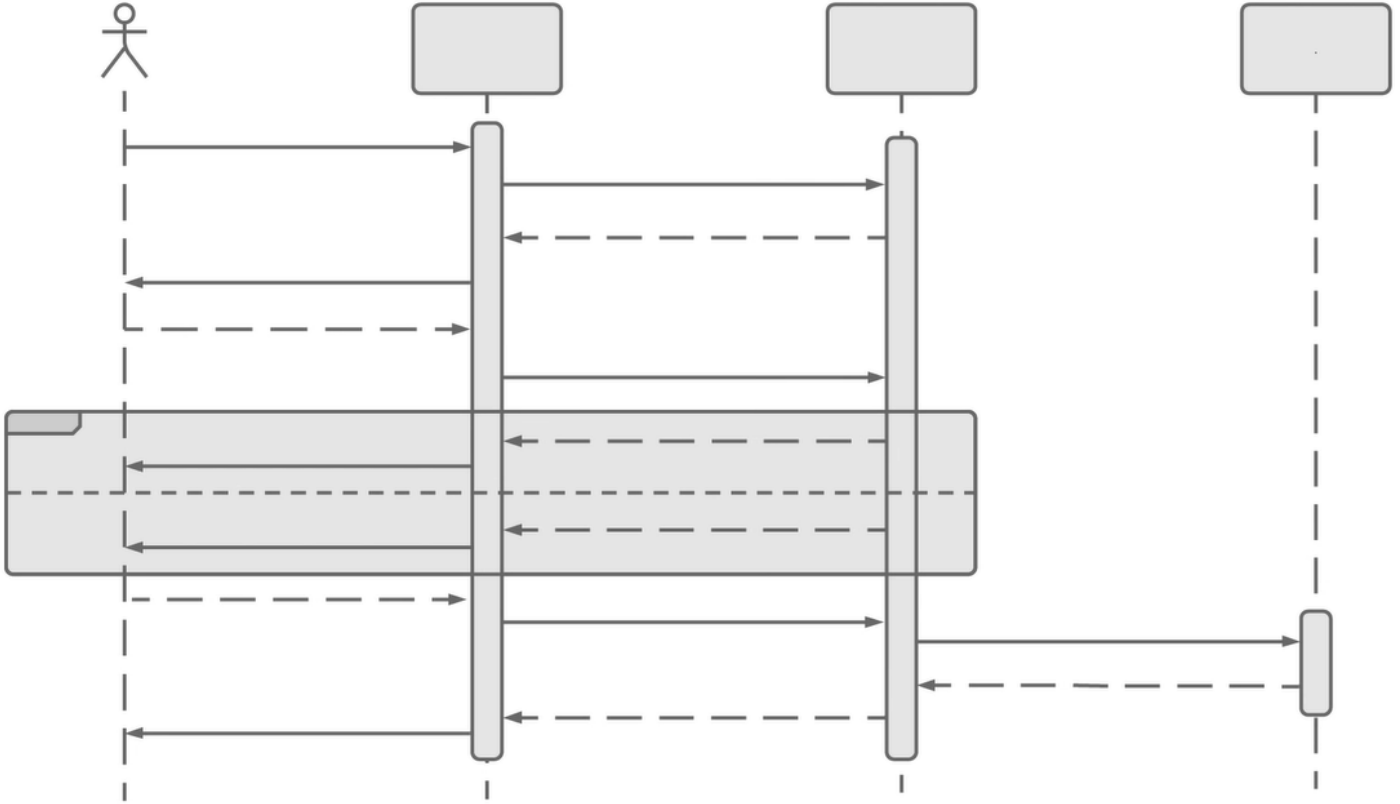

seq
Non-strict sequential message processing.
Messages are separated from each other by horizontal dashed lines and can be processed in any order except for messages received by a single object.

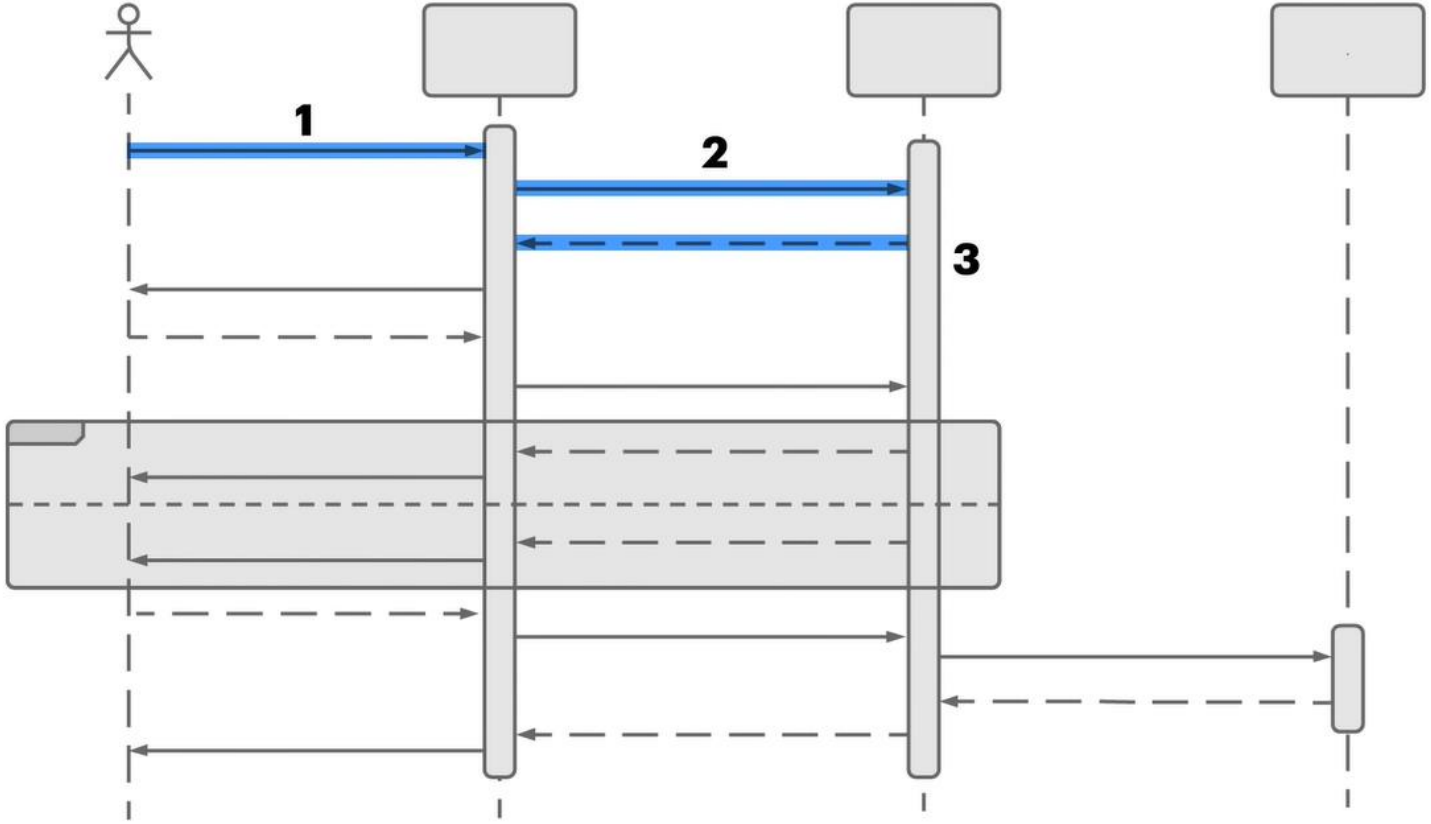

: Customer
:SearchPage
: Search Results Page
:Catalog
:Search Results
1: onSearch(author)
1.1: validateSearchCriteria()
alt
1.2: searchByAuthor(author)
1.2.1 : create()
[author entered]
1.2.1.1 : display()
[author not entered]
1.3 : displayErrorMessage()

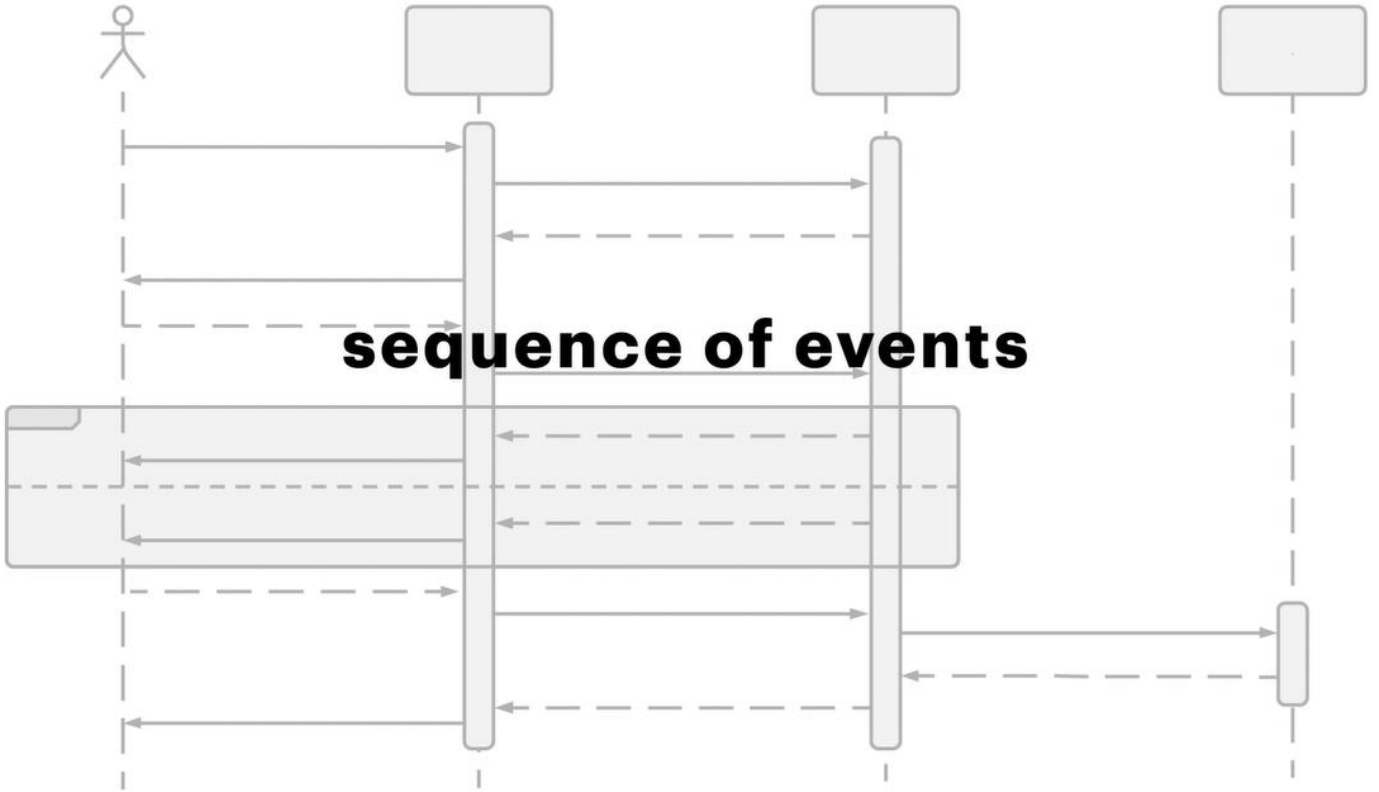
Let us see an example Sequence diagram

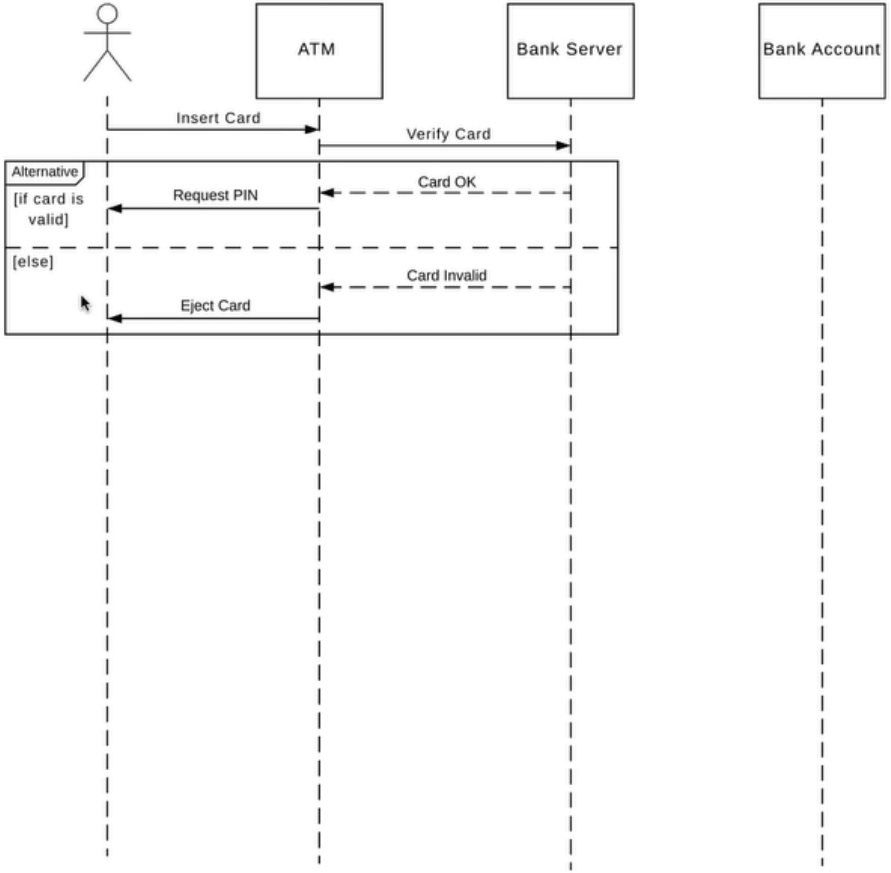
ATM Transaction

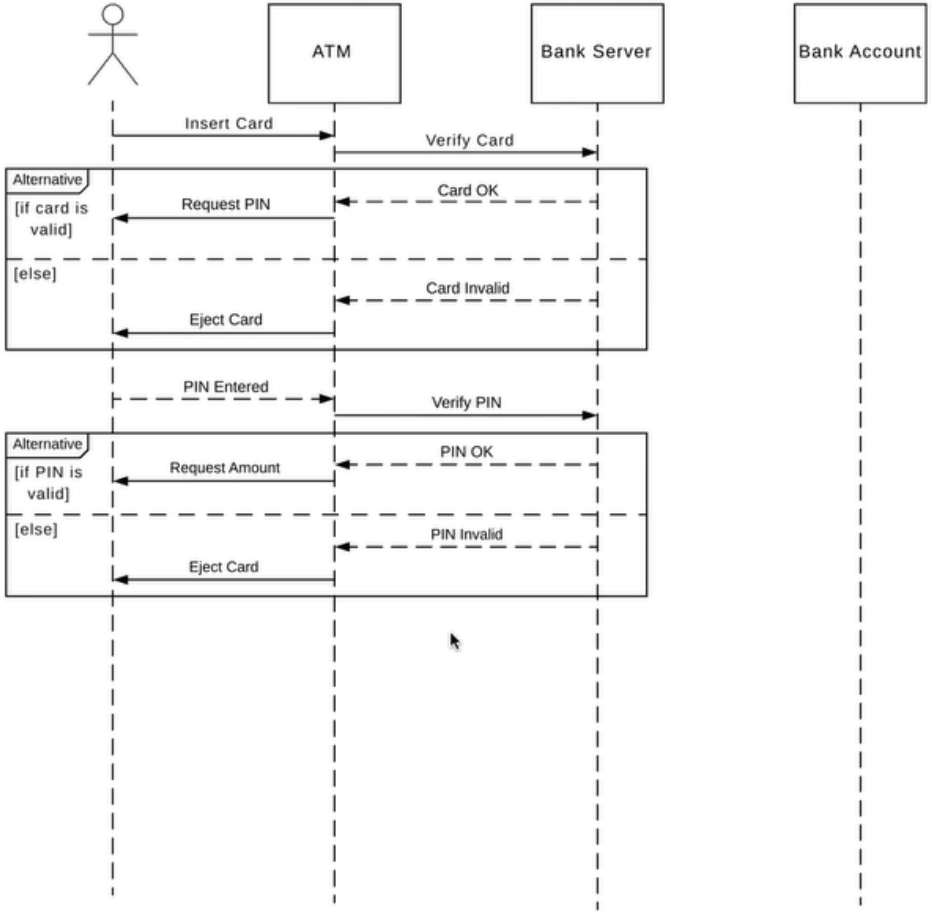

ATM

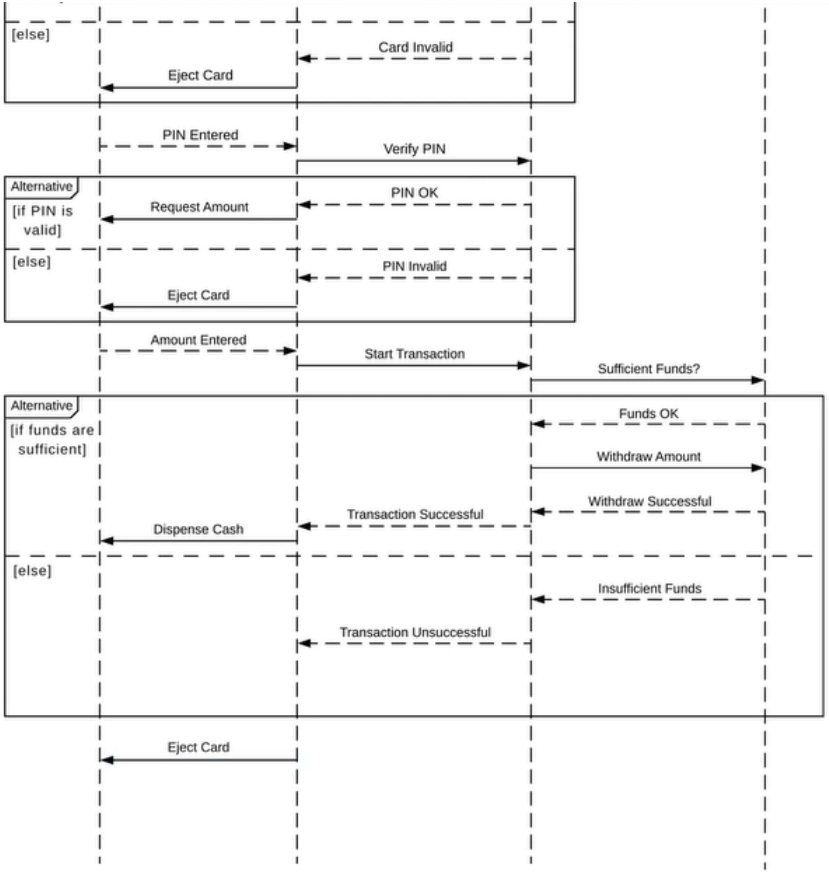


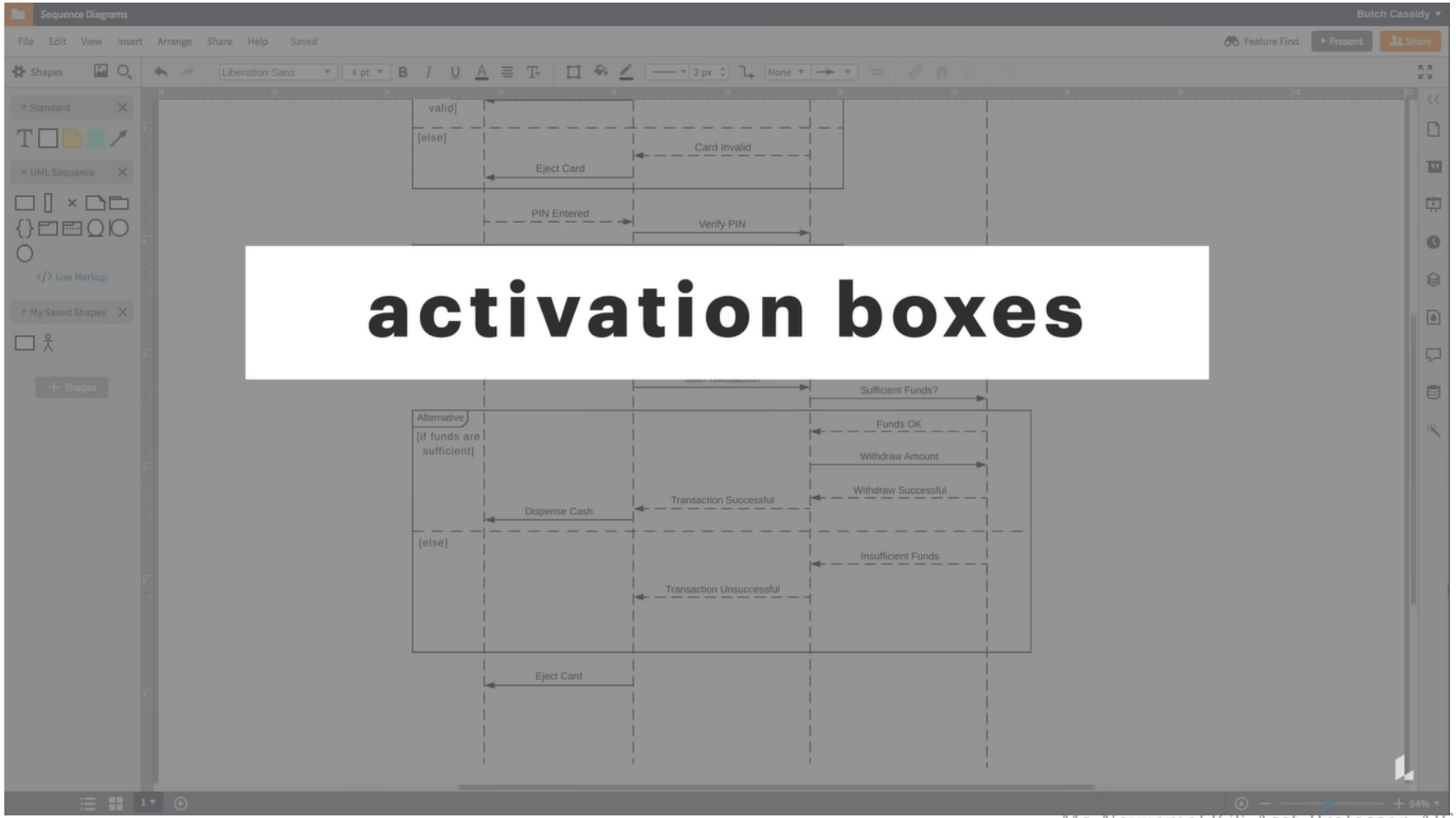

1
2
3

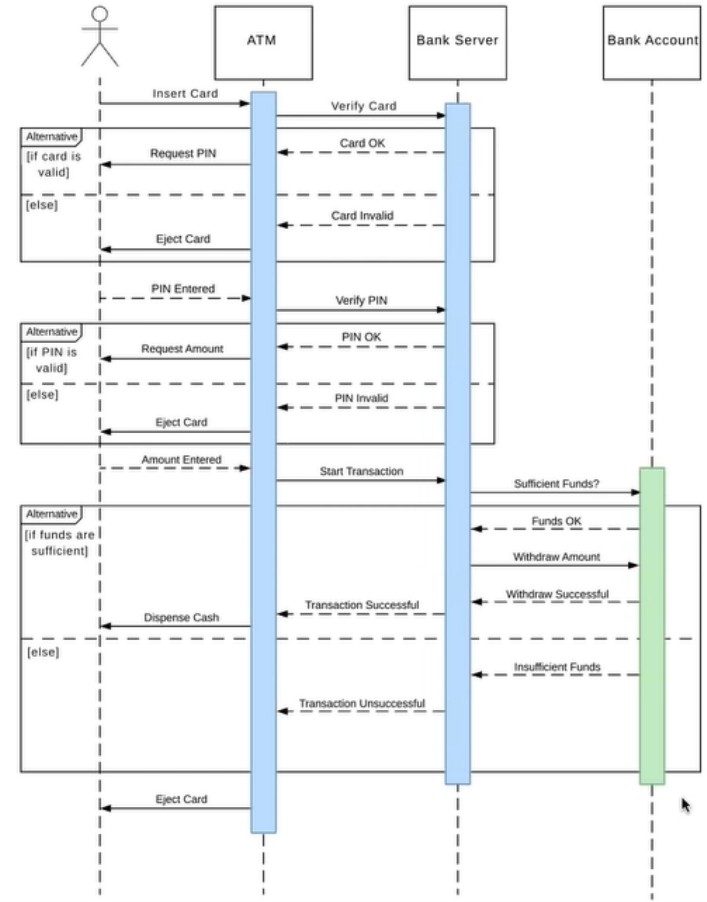

sequence of events


ATM
Bank Server
Bank Account
Insert Card
Verify Card
Alternative
[if card is valid]
Request PIN
Card OK
[else]
Card Invalid
Eject Card

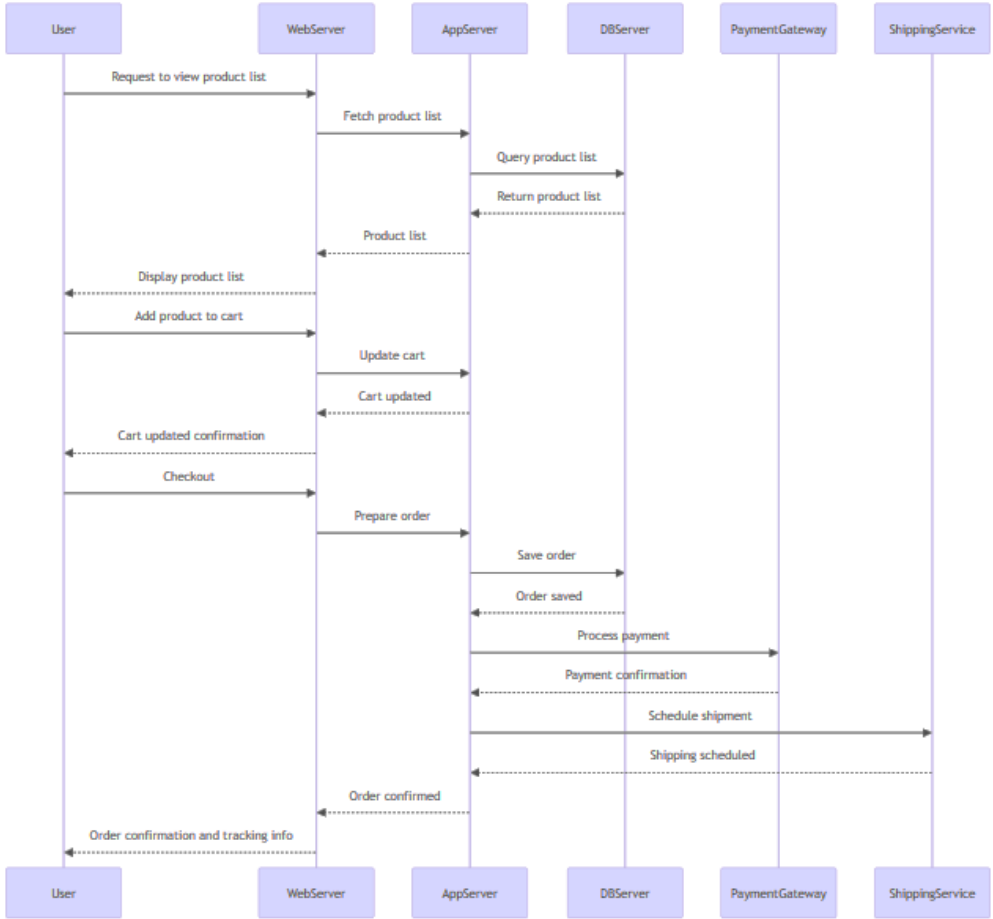

ATM
Bank Server
Bank Account
Insert Card
Verify Card
Alternative
[if card is valid]
Request PIN
Card OK
[else]
Card Invalid
Eject Card
PIN Entered
Verify PIN
Alternative
[if PIN is valid]
Request Amount
PIN OK
[else]
PIN Invalid
Eject Card


[else]
Card Invalid
Eject Card
PIN Entered
Verify PIN
Alternative
[if PIN is valid]
PIN OK
Request Amount
[else]
PIN Invalid
Eject Card
Amount Entered
Start Transaction
Sufficient Funds?
Alternative
[if funds are sufficient]
Funds OK
Withdraw Amount
Withdraw Successful
Transaction Successful
Dispense Cash
[else]
Insufficient Funds
Transaction Unsuccessful
Eject Card

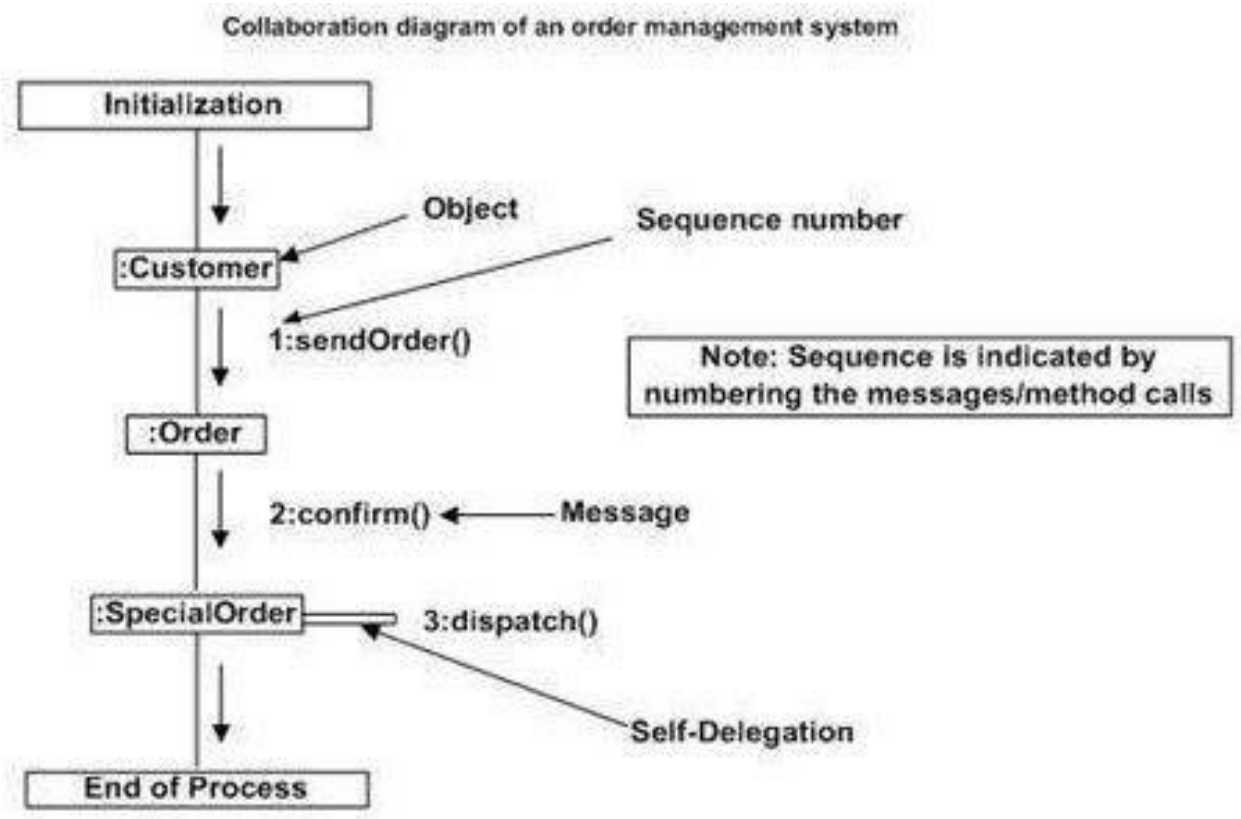

Sequence Diagrams
Butch Cassidy
File Edit View Insert Arrange Share Help Saved
Feature Find
Present
Share
Shapes
Liberation Sans
Standard
UML Sequence
Use Markup
My Saved Shapes
Shapes
valid]
[else]
Card Invalid
Eject Card
PIN Entered
Verify PIN
Sufficient Funds?
Alternative
[if funds are sufficient]
Funds OK
Withdraw Amount
Withdraw Successful
Transaction Successful
Dispense Cash
[else]
Insufficient Funds
Transaction Unsuccessful
Eject Card
activation boxes


ATM
Bank Server
Bank Account
Insert Card
Verify Card
Alternative
[if card is valid]
Request PIN
Card OK
[else]
Card Invalid
Eject Card
PIN Entered
Verify PIN
Alternative
[if PIN is valid]
Request Amount
PIN OK
[else]
PIN Invalid
Eject Card
Amount Entered
Start Transaction
Sufficient Funds?
Alternative
[if funds are sufficient]
Funds OK
Withdraw Amount
Withdraw Successful
Transaction Successful
Dispense Cash
[else]
Insufficient Funds
Transaction Unsuccessful
Eject Card

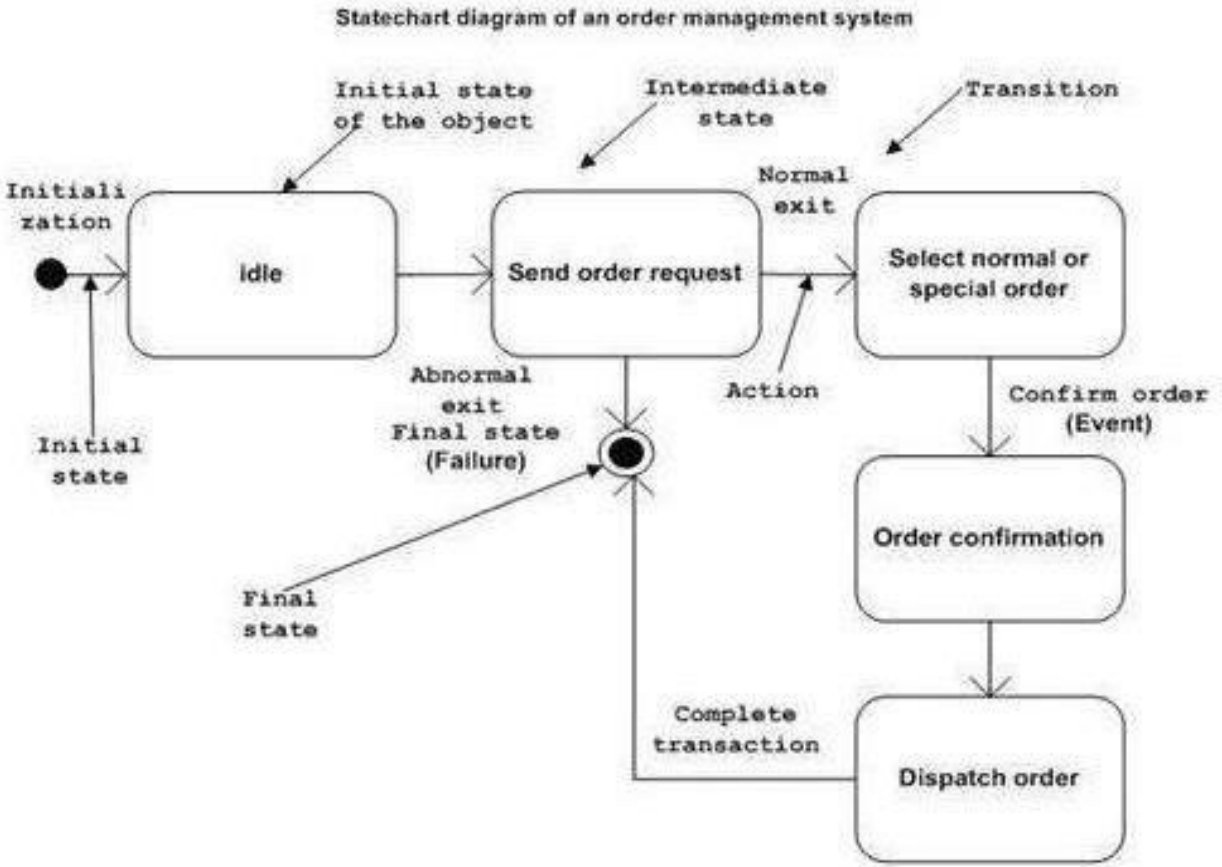
Sequence Diagram for Online Shopping Website


User
WebServer
AppServer
DBServer
PaymentGateway
ShippingService
Request to view product list
Fetch product list
Query product list
Return product list
Product list
Display product list
Add product to cart
Update cart
Cart updated
Cart updated confirmation
Checkout
Prepare order
Save order
Order saved
Process payment
Payment confirmation
Schedule shipment
Shipping scheduled
Order confirmed
Order confirmation and tracking info
User
WebServer
AppServer
DBServer
PaymentGateway
ShippingService

Collaboration diagram - Emphasizes on the structural organization of the objects that send and receive messages.

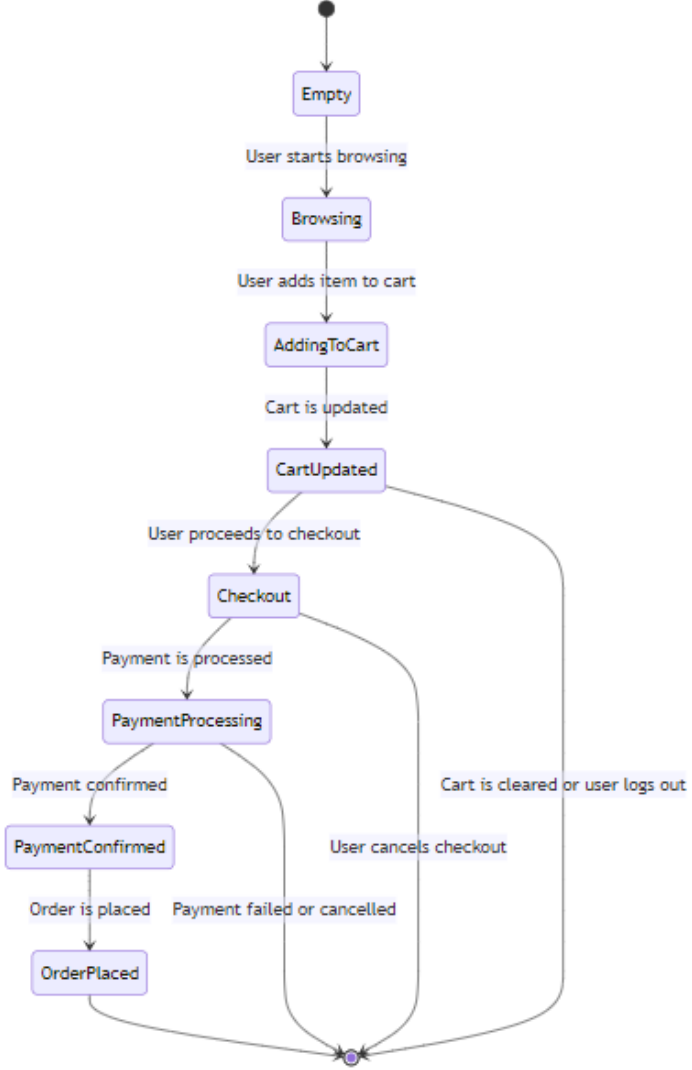


State chart Diagrams - State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.



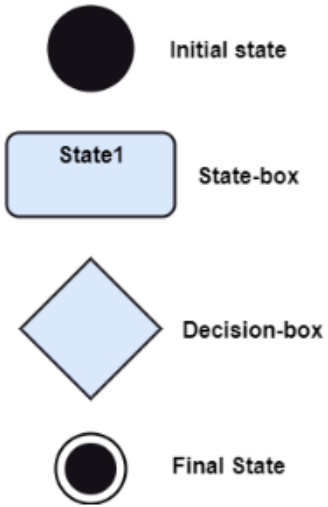
State chart Diagrams –

State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.
Example: State Chart Diagram of Online Shopping Website



State chart Diagrams

Following are the notations of a state machine diagram enlisted below:



a. **Initial state:** It defines the initial state (beginning) of a system, and it is represented by a black filled circle.

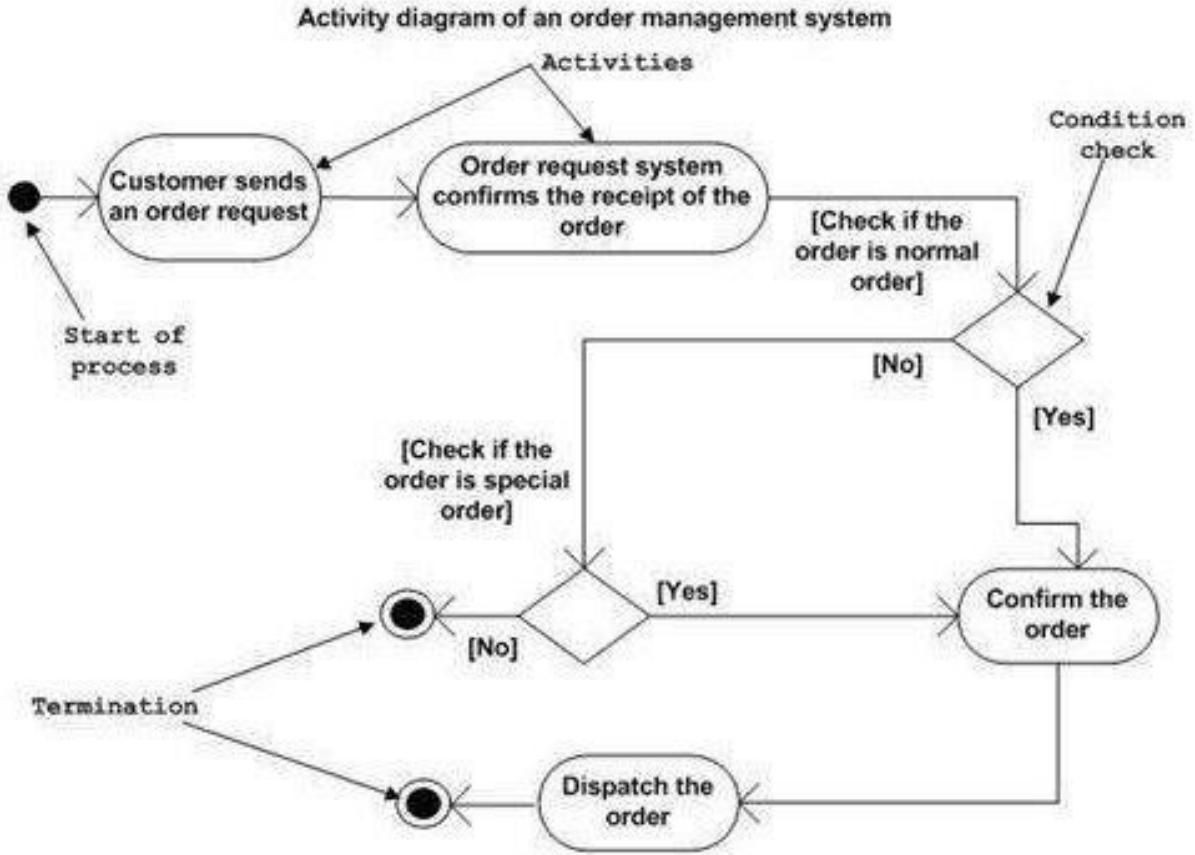
b. **Final state:** It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.

c. **Decision box:** It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.

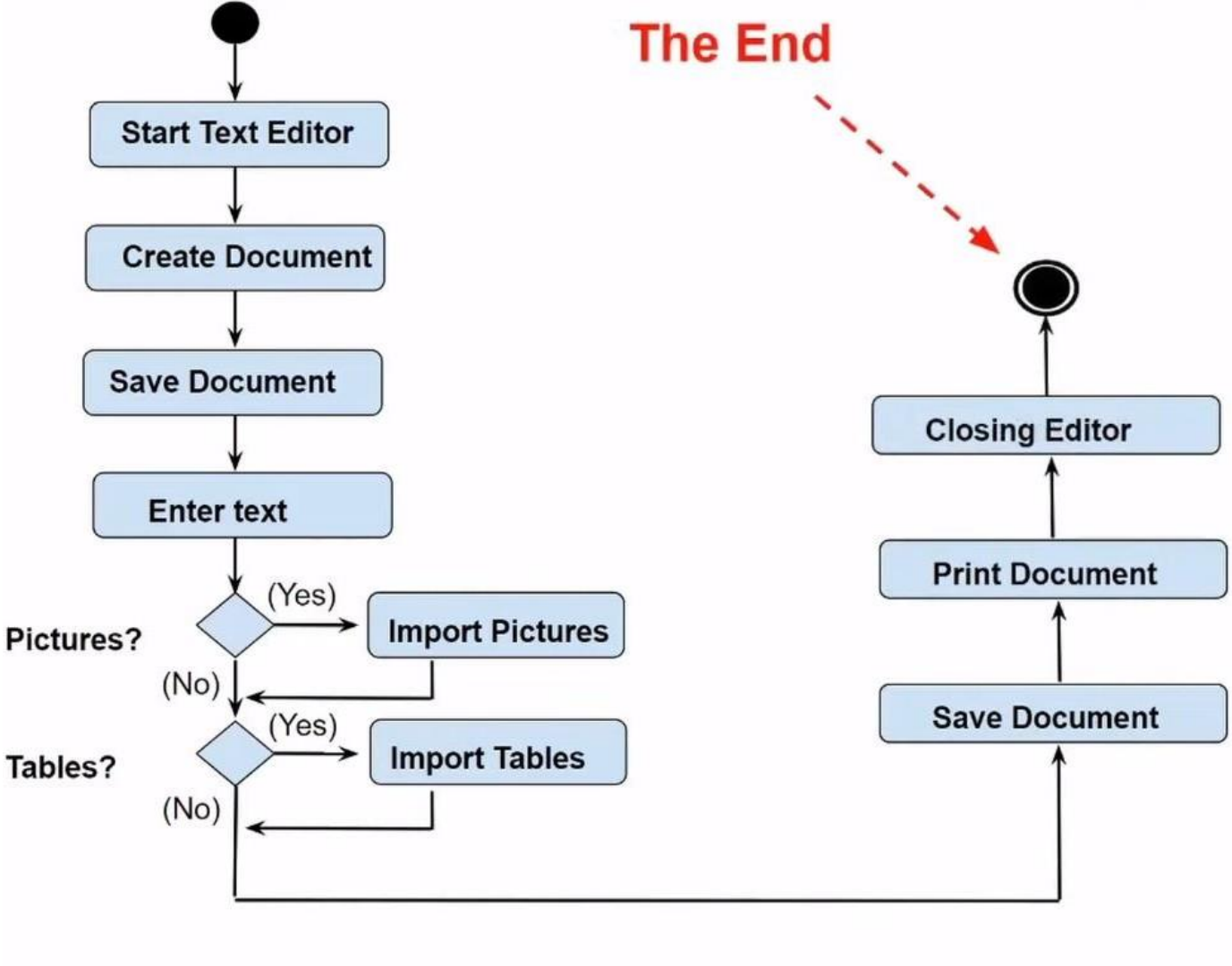
d. **Transition:** A change of control from one state to another due to the occurrence of some event is termed as a transition. It is represented by an arrow labeled with an event due to which the change has ensued.

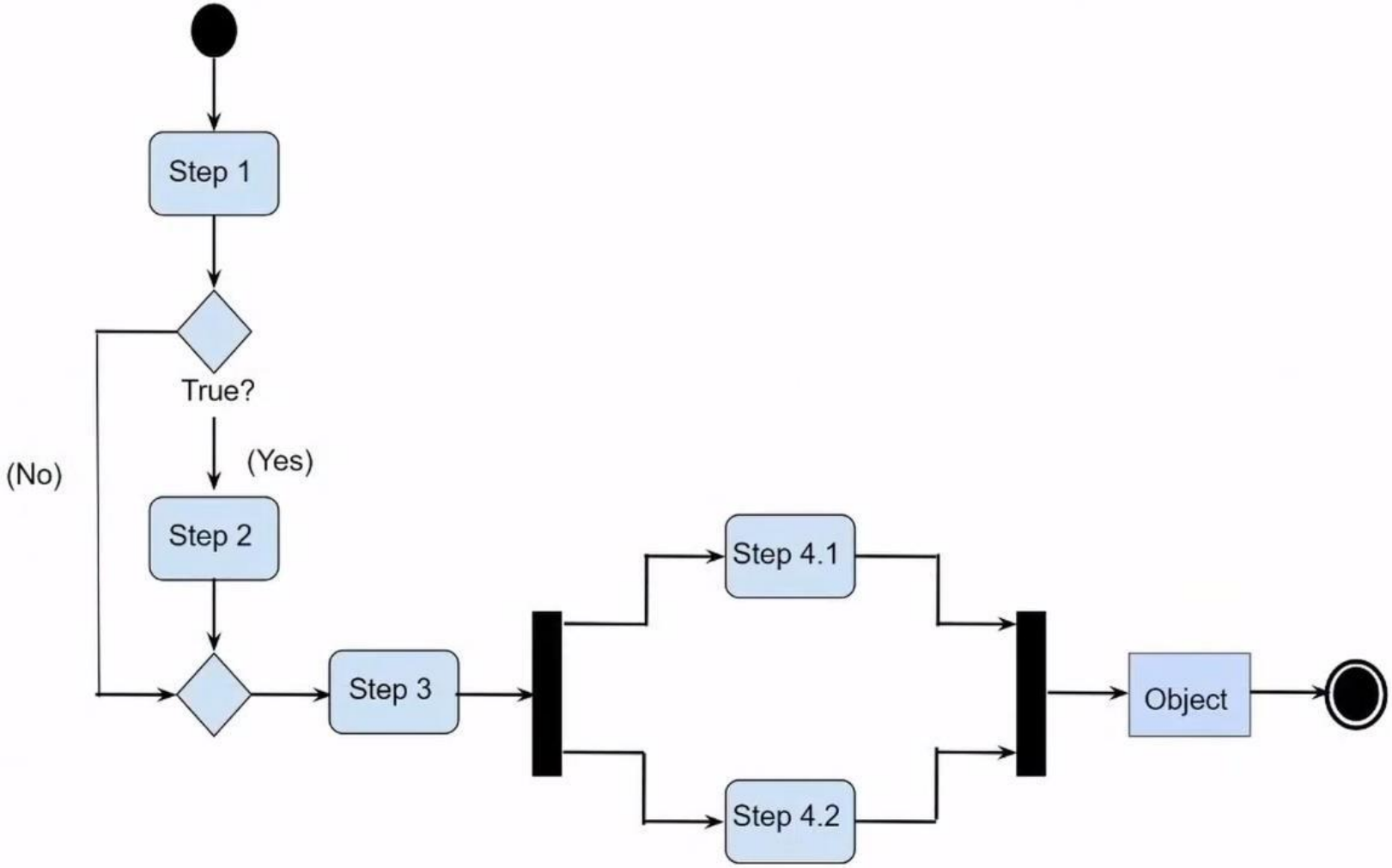
e. **State box:** It depicts the conditions or circumstances of a particular object of a class at a specific point of time. A rectangle with round corners is used to represent the state box.

Activity Diagrams – Basically a flowchart to represent the flow from one activity to another activity

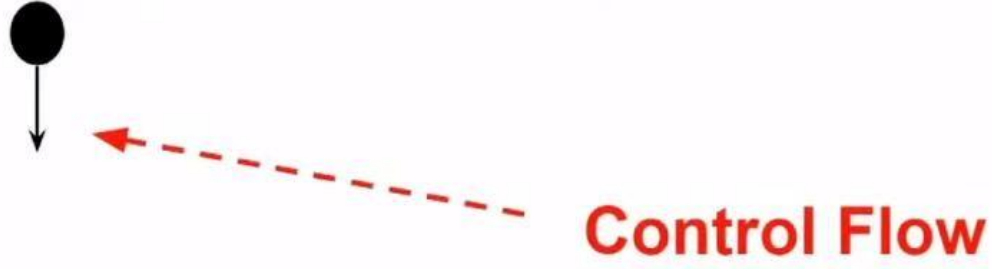


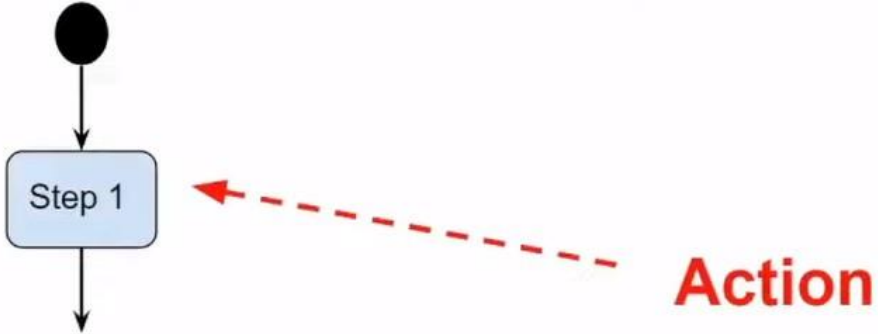
Creating a document in a text editor

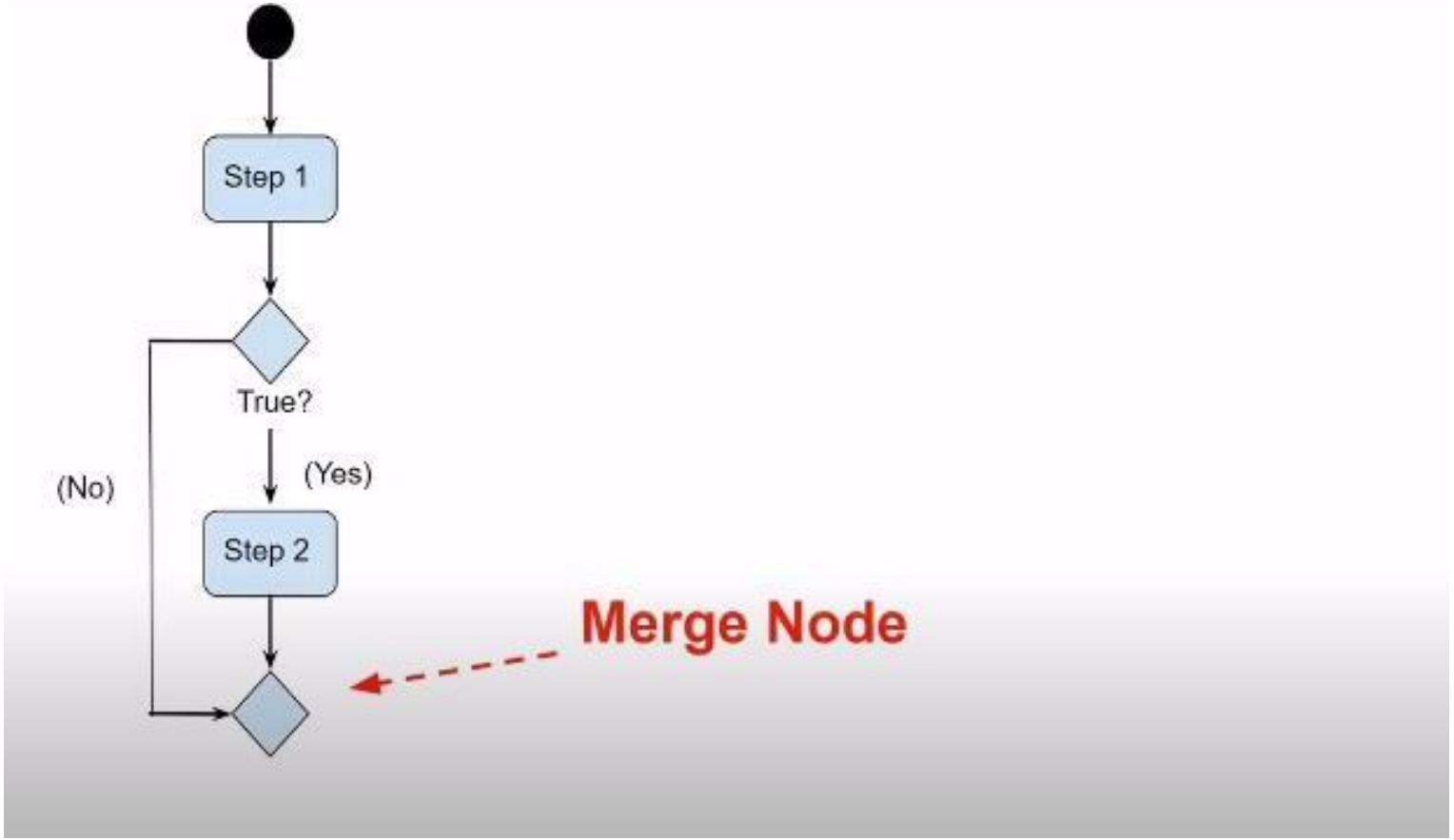

The End
Start Text Editor
Create Document
Save Document
Enter text
Pictures?
(Yes)
Import Pictures
(No)
Tables?
(Yes)
Import Tables
(No)
Closing Editor
Print Document
Save Document

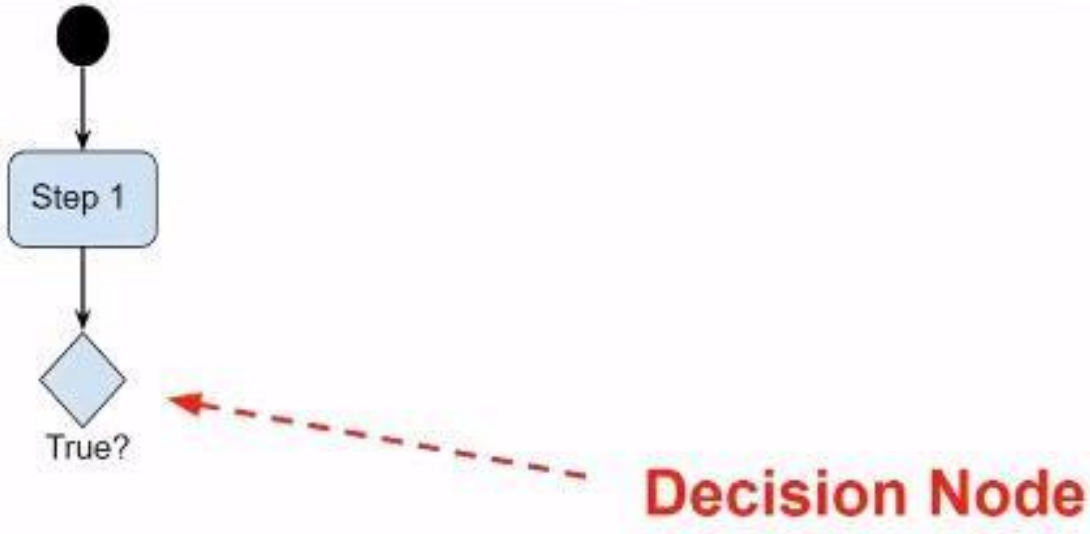

Step 1
True?
(No)
(Yes)
Step 2
Step 3
Step 4.1
Step 4.2
Object

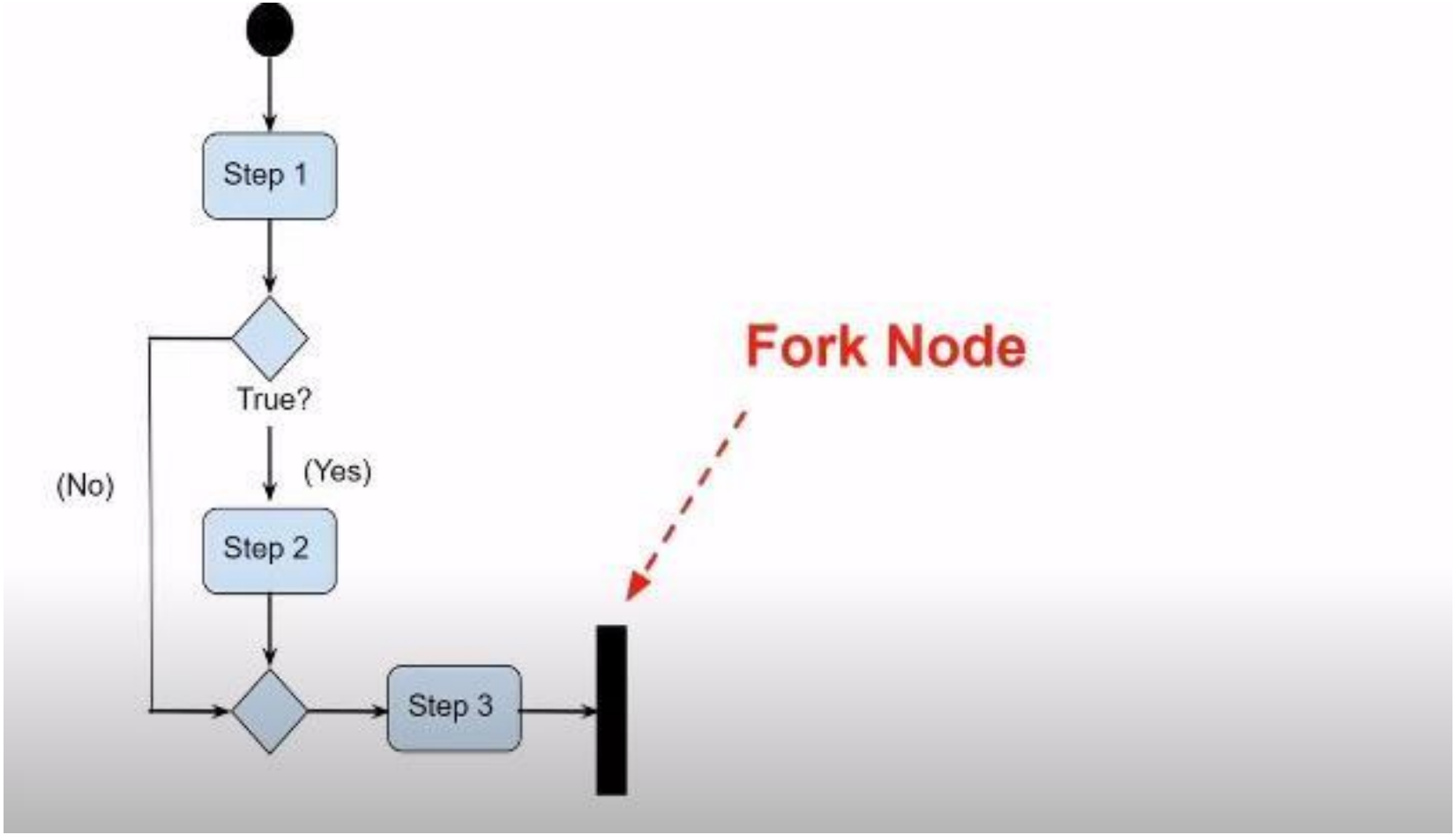

Initial Node

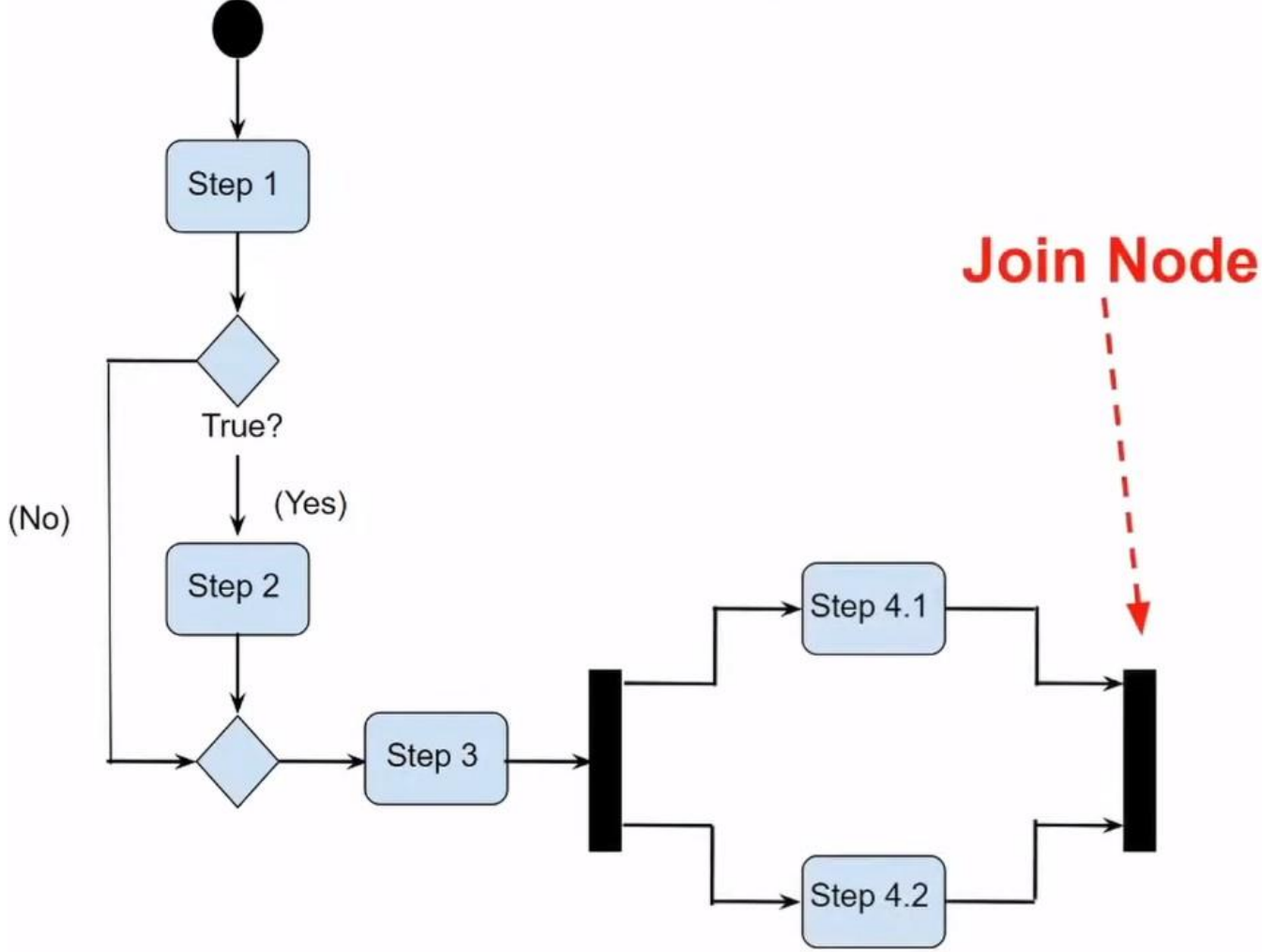

Control Flow

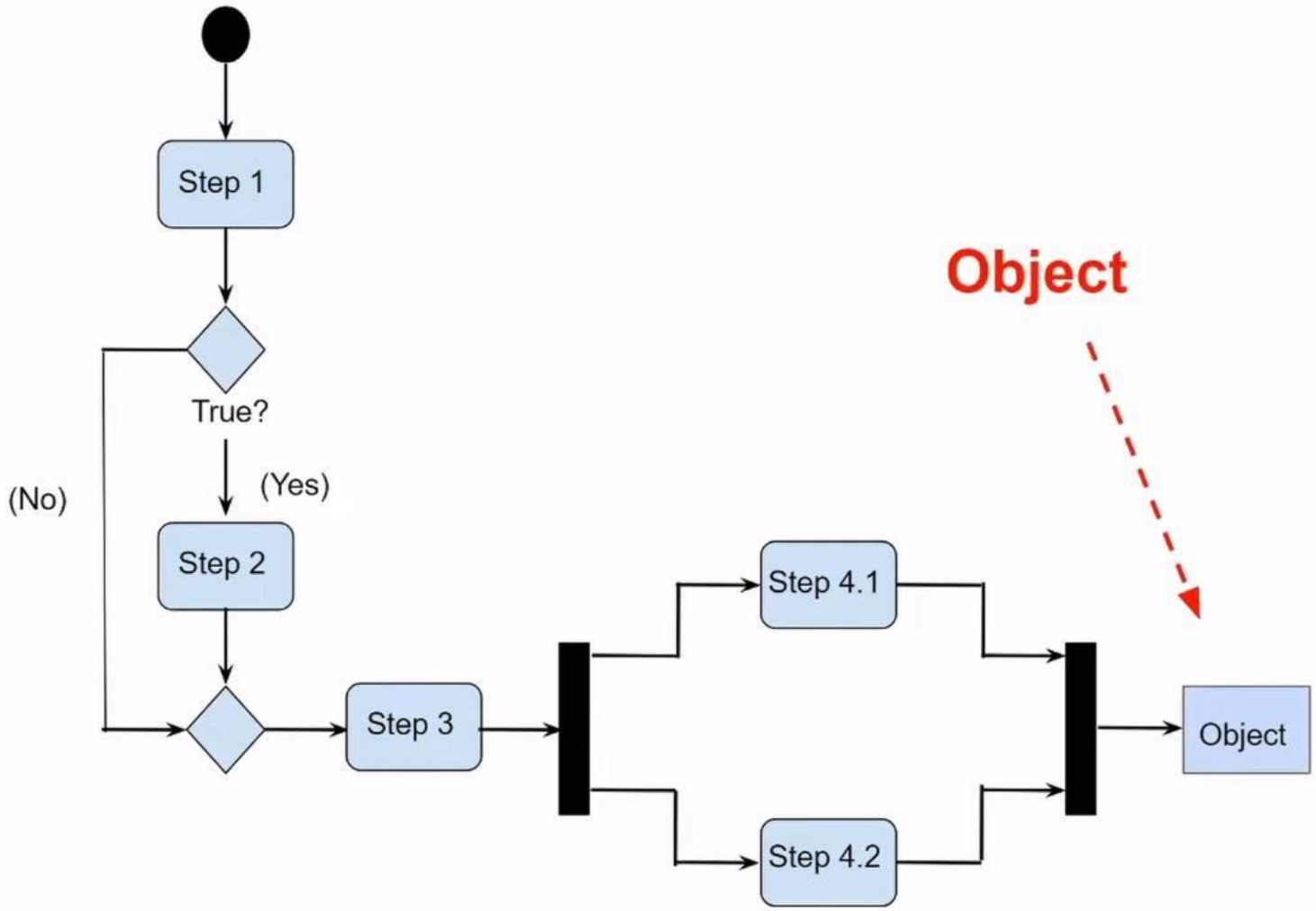

Step 1
Action


Step 1
True?
(No)
(Yes)
Step 2
Merge Node

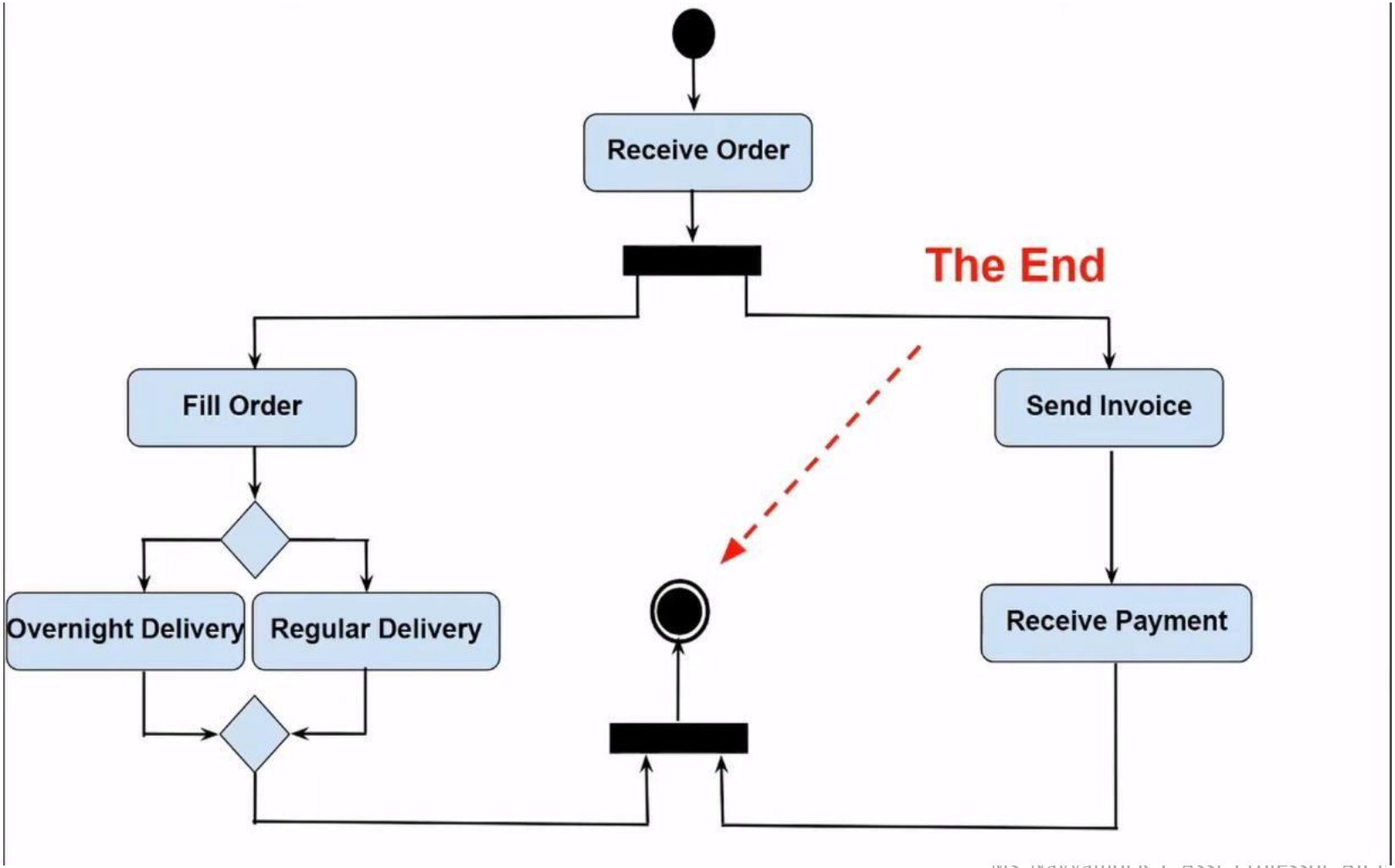

Step 1
True?
Decision Node

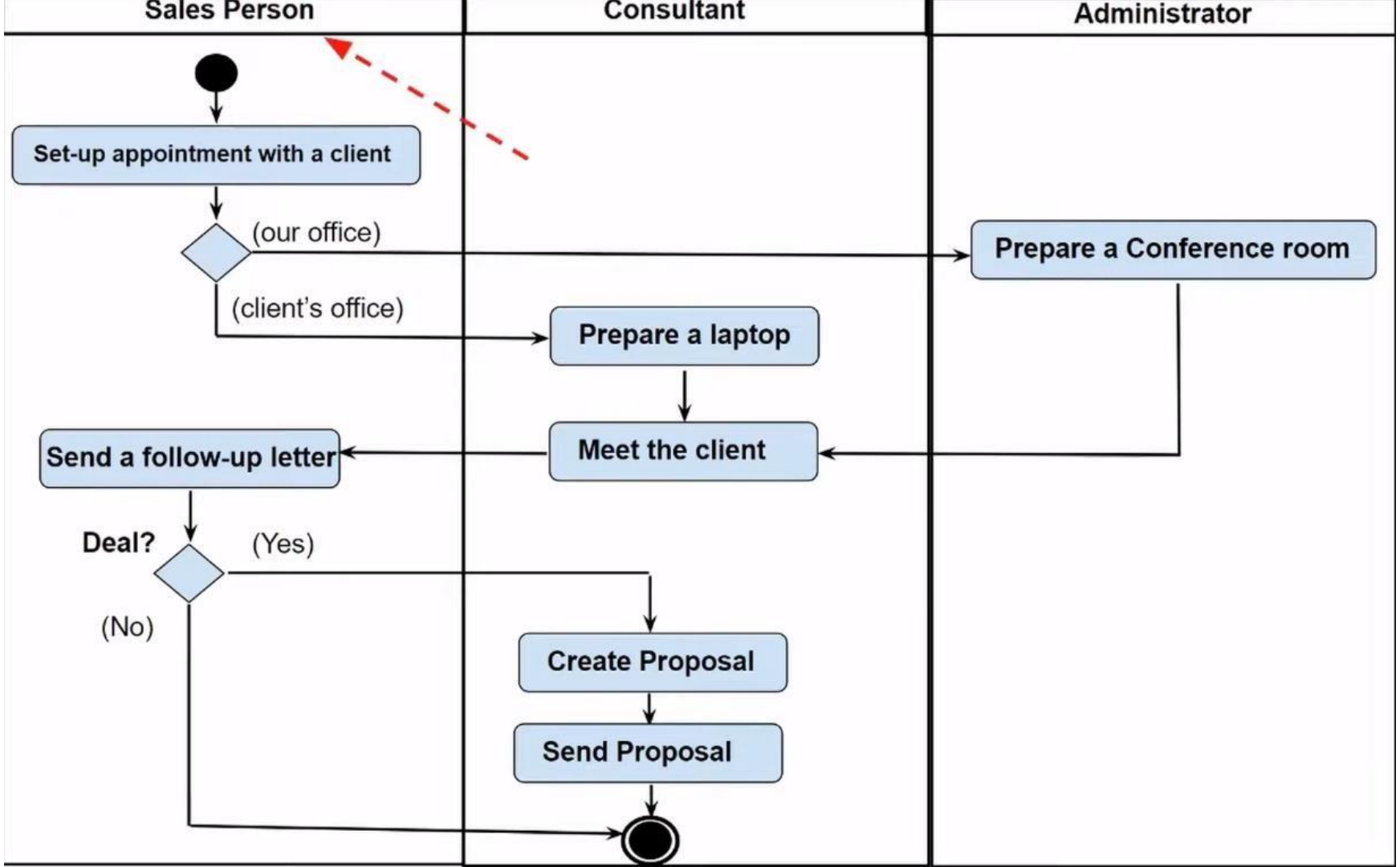

Step 1
True?
(No)
(Yes)
Step 2
Step 3
Fork Node


Step 1
True?
(No)
(Yes)
Step 2
Step 3
Step 4.1
Step 4.2
Join Node

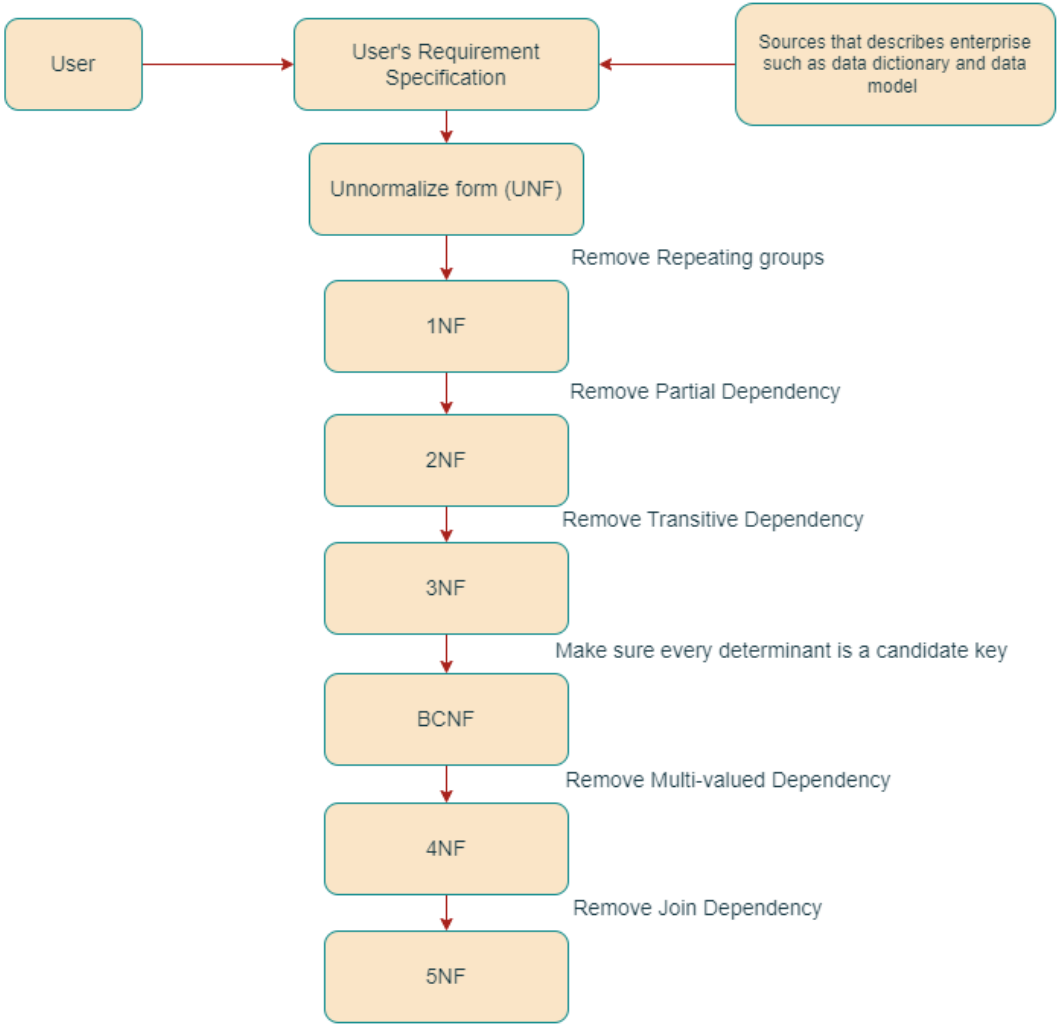

Step 1
True?
(No)
(Yes)
Step 2
Step 3
Step 4.1
Step 4.2
Object
Object

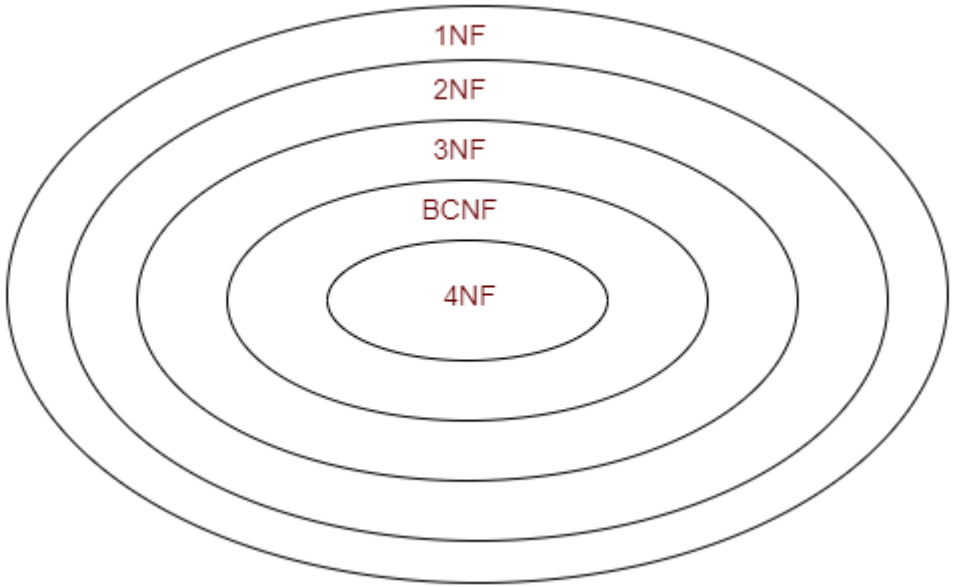
Order Processing

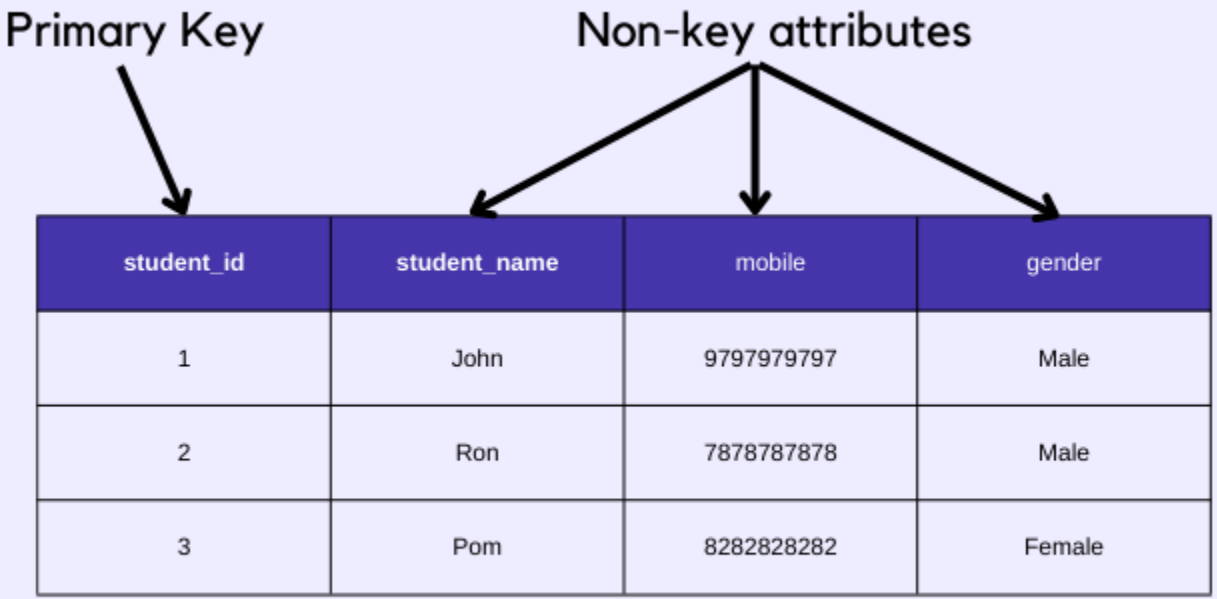

Receive Order
The End
Fill Order
Send Invoice
Overnight Delivery
Regular Delivery
Receive Payment


Sales Person
Consultant
Administrator
Set-up appointment with a client
(our office)
(client's office)
Prepare a Conference room
Prepare a laptop
Meet the client
Send a follow-up letter
Deal?
(Yes)
(No)
Create Proposal
Send Proposal

NORMALIZATION


User
User's Requirement Specification
Sources that describes enterprise such as data dictionary and data model
Unnormalize form (UNF)
Remove Repeating groups
1NF
Remove Partial Dependency
2NF
Remove Transitive Dependency
3NF
Make sure every determinant is a candidate key
BCNF
Remove Multi-valued Dependency
4NF
Remove Join Dependency
5NF


1NF
2NF
3NF
BCNF
4NF


Primary Key
Non-key attributes
student_id
student_name
mobile
gender
1
John
9797979797
Male
2
Ron
7878787878
Male
3
Pom
8282828282
Female

What is Normalization?

- Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization rules divides larger tables into smaller tables and links them using relationships.
- The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

1ST NORMAL FORM

Removes repeating groups from the table

Create a separate table for each set of related data

Identify each set of related data with a primary key

Consider a Table Employee

Employee ID	Employee Name	Phone Number	Salary
1EDU001	Alex	+91 8553206126 +91 9449424949	60,131
1EDU002	Barry	+91 8762989672	48,302
1EDU003	Clair	+91 9916255225	22,900
1EDU004	David	+91 6363625811 +91 8762055007	81,538

1 NF EXAMPLE

Employee ID	Employee Name	Phone Number	Salary
1EDU001	Alex	+91 8553206126	60,131
1EDU001	Alex	+91 9449424949	60,131
1EDU002	Barry	+91 8762989672	48,302
1EDU003	Clair	+91 9916255225	22,900
1EDU004	David	+91 6363625811	81,538
1EDU004	David	+91 8762055007	81,538

2nd NORMAL FORM

It has to be in 1st Normal Form

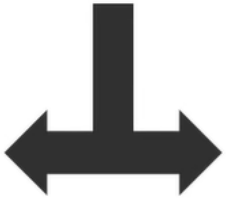
Table also should not contain partial dependency

Employee Id	Department Id	Office Location
1EDU001	ED-T1	Pune
1EDU002	ED-S2	Bengaluru
1EDU003	ED-M1	Delhi
1EDU004	ED-T3	Mumbai

2 NF EXAMPLE

Employee Id	Department Id	Office Location
1EDU001	ED-T1	Pune
1EDU002	ED-S2	Bengaluru
1EDU003	ED-M1	Delhi
1EDU004	ED-T3	Mumbai

Employee Id	Department Id
1EDU001	ED-T1
1EDU002	ED-S2
1EDU003	ED-M1
1EDU004	ED-T3



Department Id	Office Location
ED-T1	Pune
ED-S2	Bengaluru
ED-M1	Delhi
ED-T3	Mumbai

3rd NORMAL FORM

It has to be in 2nd Normal Form

There should be no transitive dependency for non-prime attributes

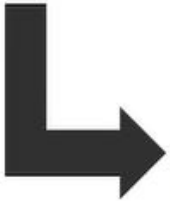
3rd NORMAL FORM

Student Id	Student Name	Subject Id	Subject	Address
1DT15ENG01	Alex	15CS11	SQL	Goa
1DT15ENG02	Barry	15CS13	JAVA	Bengaluru
1DT15ENG03	Clair	15CS12	C++	Delhi
1DT15ENG04	David	15CS13	JAVA	Kochi

Student Id	Student Name	Subject Id	Subject	Address
1DT15ENG01	Alex	15CS11	SQL	Goa
1DT15ENG02	Barry	15CS13	JAVA	Bengaluru
1DT15ENG03	Clair	15CS12	C++	Delhi
1DT15ENG04	David	15CS13	JAVA	Kochi

Student Id	Student Name	Subject Id	Address
1DT15ENG01	Alex	15CS11	Goa
1DT15ENG02	Barry	15CS13	Bengaluru
1DT15ENG03	Clair	15CS12	Delhi
1DT15ENG04	David	15CS13	Kochi

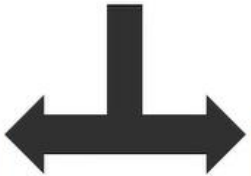
Student Id	Student Name	Subject Id	Subject	Address
1DT15ENG01	Alex	15CS11	SQL	Goa
1DT15ENG02	Barry	15CS13	JAVA	Bengaluru
1DT15ENG03	Clair	15CS12	C++	Delhi
1DT15ENG04	David	15CS13	JAVA	Kochi



Subject Id	Subject
15CS11	SQL
15CS13	JAVA
15CS12	C++
15CS13	JAVA

3rd NORMAL FORM

Student Id	Student Name	Subject Id	Subject	Address
1DT15ENG01	Alex	15CS11	SQL	Goa
1DT15ENG02	Barry	15CS13	JAVA	Bengaluru
1DT15ENG03	Clair	15CS12	C++	Delhi
1DT15ENG04	David	15CS13	JAVA	Kochi



Student Id	Student Name	Subject Id	Address
1DT15ENG01	Alex	15CS11	Goa
1DT15ENG02	Barry	15CS13	Bengaluru
1DT15ENG03	Clair	15CS12	Delhi
1DT15ENG04	David	15CS13	Kochi

Subject Id	Subject
15CS11	SQL
15CS13	JAVA
15CS12	C++
15CS13	JAVA

BC NORMAL FORM

It has to be in 3rd Normal Form

Higher version 3NF and was developed by Raymond F. Boyce and Edgar F. Codd

Every functional dependency A → B, then A has to be the Super Key of that particular table

BC NORMAL FORM

Student ID	Subject	Professor
1DT15ENG01	SQL	Prof. Mishra
1DT15ENG02	JAVA	Prof. Anand
1DT15ENG02	C++	Prof. Kanthi
1DT15ENG03	JAVA	Prof. Anand
1DT15ENG04	DBMS	Prof. Lokesh

BCNF EXAMPLE

Student ID	Professor ID
1DT15ENG01	1DTPF01
1DT15ENG02	1DTPF02
1DT15ENG02	1DTPF03
1DT15ENG03	1DTPF02
1DT15ENG04	1DTPF04

Professor ID	Subject	Professor
1DTPF01	SQL	Prof. Mishra
1DTPF02	JAVA	Prof. Anand
1DTPF03	C++	Prof. Kanthi
1DTPF02	JAVA	Prof. Anand
1DTPF04	DBMS	Prof. Lokesh

Guidelines for Normalizing Tables

- **For Relational Databases (MySQL, MS SQL)**
- Normalization is the process of organizing a database to minimize redundancy and improve data integrity. Follow these steps to normalize tables:

1. **First Normal Form (1NF):**
   - Ensure that each table has a primary key and that all columns contain atomic (indivisible) values.
   - **Guideline:** Eliminate repeating groups or arrays. Each column must contain only one value per row.
2. **Second Normal Form (2NF):**
   - Ensure that all non-key attributes are fully functionally dependent on the primary key.
   - **Guideline:** Remove partial dependencies. If a table has a composite primary key, ensure that all non-key attributes are dependent on the entire key, not just part of it.
3. **Third Normal Form (3NF):**
   - Ensure that all attributes are only dependent on the primary key and not on other non-key attributes.
   - **Guideline:** Remove transitive dependencies. Each non-key attribute should be directly dependent on the primary key.

Guidelines for Normalizing Tables

- **For Object-Oriented Databases**
- Normalization in object-oriented databases focuses on reducing redundancy and ensuring proper encapsulation of data. The concept differs slightly from relational normalization:

1. **Encapsulation:**
   - Ensure that each object class encapsulates its own data and methods.
   - **Guideline:** Avoid exposing internal data structures. Use methods to interact with object data.
2. **Inheritance:**
   - Use inheritance to handle common attributes and methods.
   - **Guideline:** Avoid duplication by creating base classes with common features
3. **Association and Aggregation:**
   - Ensure that relationships between objects are well-defined and managed.
   - **Guideline:** Use association and aggregation to model relationships between objects, reflecting their real-world connections.
4. **Polymorphism:**
   - Use polymorphism to handle different types of objects through a common interface.
   - **Guideline:** Implement polymorphism to allow objects of different classes to be treated through a common interface.

INDEXING

Indexing in Relational Schema

- **Purpose:** Indexing improves the speed of data retrieval operations on a database by creating an index (a data structure) that allows the database to find rows more quickly.
- **Types of Indexes:**
  - **Primary Index:** Automatically created for primary keys. Ensures uniqueness and speed in queries involving the primary key.
  - **Secondary Index:** Created on columns that are frequently searched but are not primary keys.
  - **Composite Index:** Created on multiple columns to speed up queries that filter on multiple fields.
  - Example : CREATE INDEX idx_customer_name ON Customers (customerName);

Best Practices:

- **Index Only Frequently Queried Columns:** Avoid creating indexes on columns that are rarely used in queries.
- **Avoid Excessive Indexing:** Too many indexes can slow down write operations (INSERT, UPDATE, DELETE).
- **Regular Maintenance:** Periodically review and optimize indexes based on query performance.

Indexing in MongoDB

- **Types of Indexes:**
  - **Single Field Index:** Indexes a single field in a collection.
  - **Compound Index:** Indexes multiple fields in a collection.
  - **Text Index:** Allows for text search on string fields.
  - **Geospatial Index:** Supports querying of geospatial data.
  - **TTL Index:** Automatically removes documents after a certain time period.
- **Creating Indexes in MogoDB:**
- // Single Field Index
- db.collection.createIndex({ field_name: 1 }); // 1 for ascending, -1 for descending
- // Compound Index
- db.collection.createIndex({ field1: 1, field2: -1 });

- **Impact on Performance:**
- Speed Up Searches: Indexes significantly improve query performance for SELECT statements.

- **Slower Writes:**
- Indexes can slow down data insertion and updates due to the overhead of maintaining the index.

DATA SANITIZATION

Ms. Navyamol K T, Asst. Professor, AJCE

Data Sanitization

- Sanitization ensures that data input by users is clean and secure, preventing SQL injection and other types of attacks.
- Many web applications receive external input. External input/data can be:
- User input from a form, Cookies, Web services data, Server Variables, Database query results
- **Input Validation:**
  - **Validate Input:** Ensure that data conforms to expected formats (e.g., email addresses, phone numbers).
  - **SQL Injection Prevention:** Use parameterized queries or prepared statements to handle user input safely.

Data Sanitization

- **Sanitizing a String:** The following example uses the **filter_var()** function to remove all HTML tags from a string in php.

```php
<?php

//Use of filter_var()
$str = "<h1>Amal Jyothi College</h1>";

$newstr = filter_var($str,
    FILTER_SANITIZE_STRING);

echo $newstr;
?>
```

Additional Data base things you can implement in your Project

Stored Procedures and Triggers:

- Predefined SQL code that can be executed with a single call.
- **Triggers:** Automatically execute a specified action in response to certain events (e.g., INSERT, UPDATE, DELETE).

Backup and Recovery:

- Regularly create backups and test recovery procedures.

Data Encryption:

- Protects sensitive data by encoding it.
- **Implementation:** Use built-in database encryption features or external encryption tools.

Aggregation Framework:

- Perform complex data analysis and transformations within MongoDB.
- **Implementation:** Use aggregation pipelines to filter, group, and sort data.

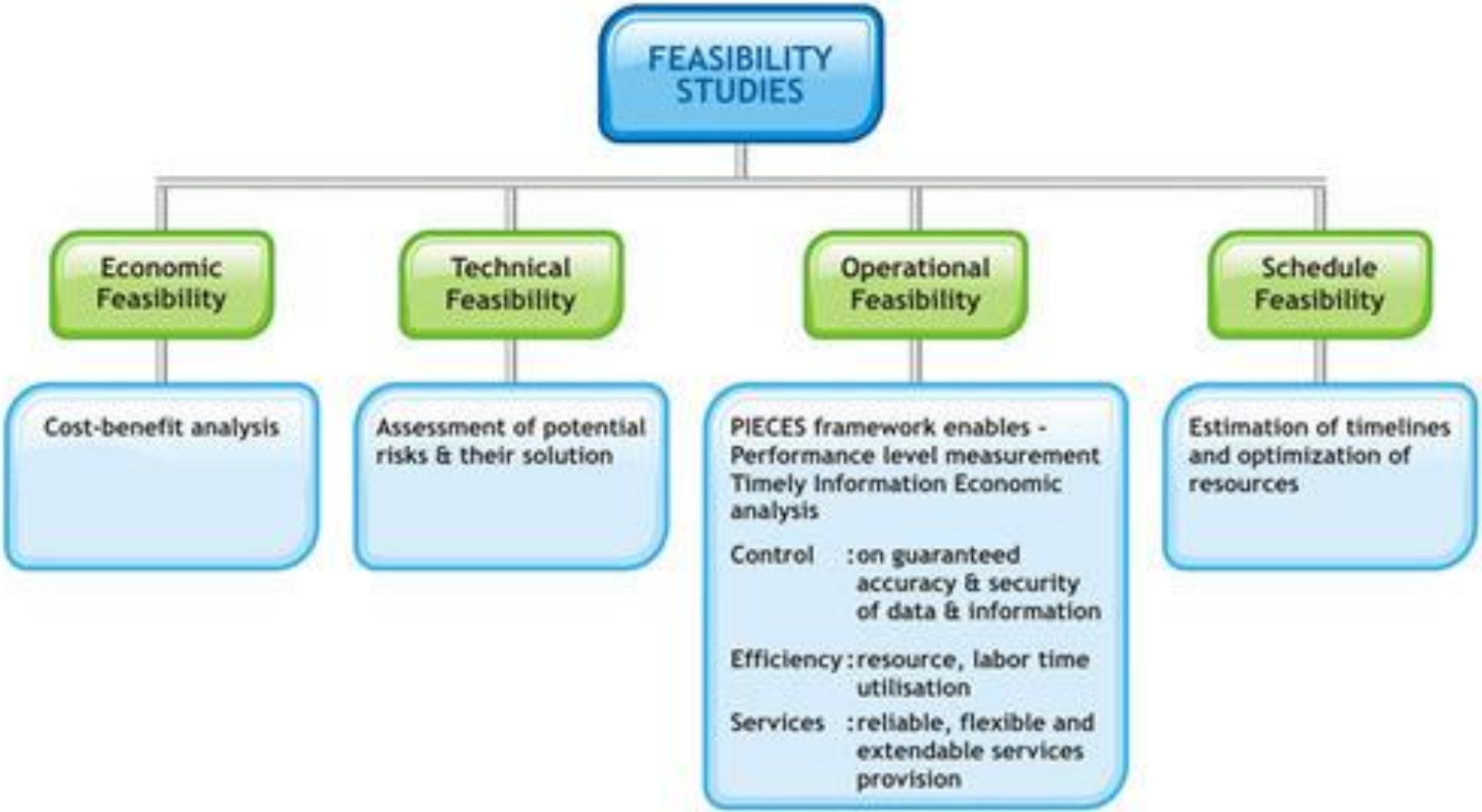
Replication and Sharding:

- **Replication:** Ensures high availability by duplicating data across multiple servers.
- **Sharding:** Distributes data across multiple servers to handle large volumes of data.


FEASIBILITY
STUDY
IN PROJECT MANAGEMENT

Feasibility Study

- A **feasibility study** looks at a set of criteria in order to provide decision makers with a recommendation, whether a specific solution is feasible and viable in a certain context or not.

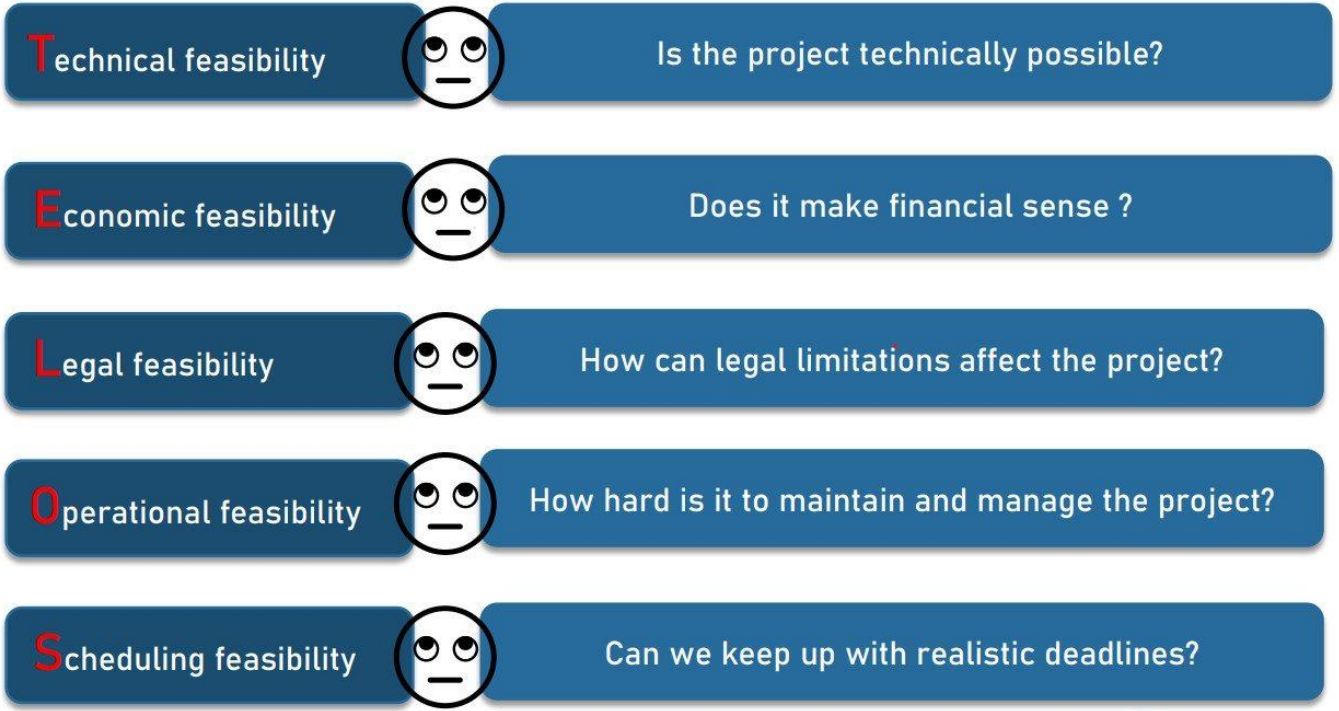

FEASIBILITY STUDIES
Economic Feasibility
Technical Feasibility
Operational Feasibility
Schedule Feasibility
Cost-benefit analysis
Assessment of potential risks & their solution
PIECES framework enables - Performance level measurement Timely Information Economic analysis
Control : on guaranteed accuracy & security of data & information
Efficiency : resource, labor time utilisation
Services : reliable, flexible and extendable services provision
Estimation of timelines and optimization of resources

Types of Feasibility

- Technical Feasibility
- Economic Feasibility
- It refers to the analysis of the cost-effectiveness of a project in order to determine whether the company should undertake the project on the basis of profitability or not.
- Example
- Operational Feasibility

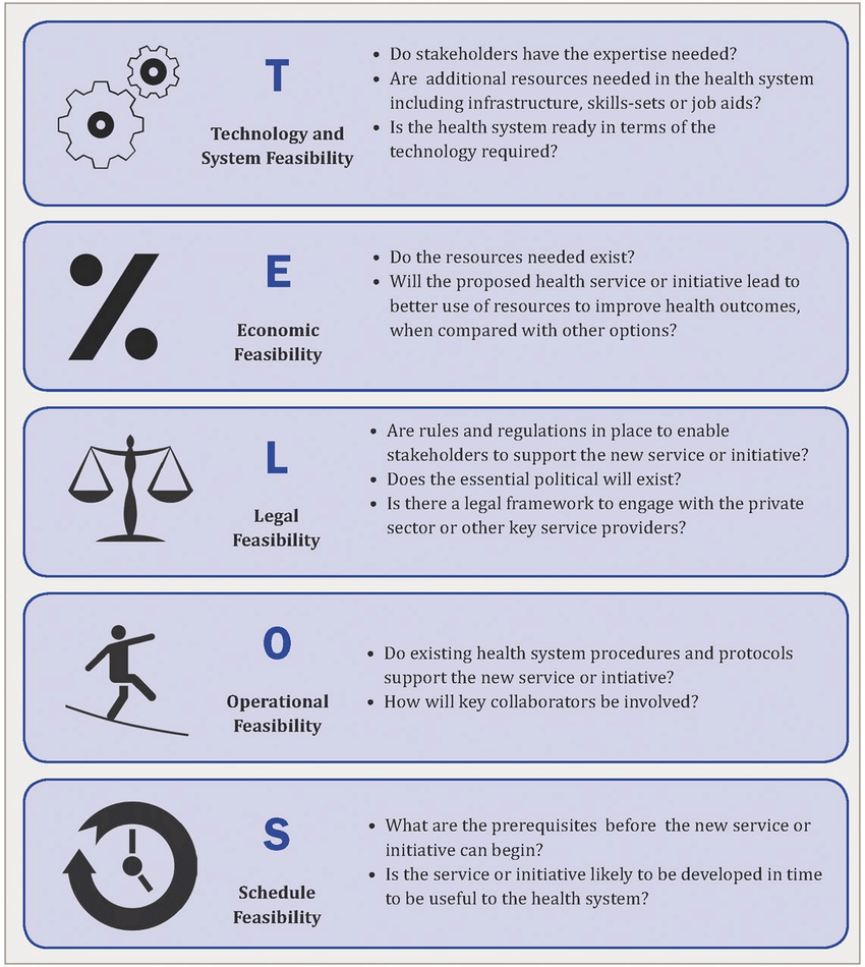
Technical Feasibility

TELOS framework



Feasibility Study Format for Project

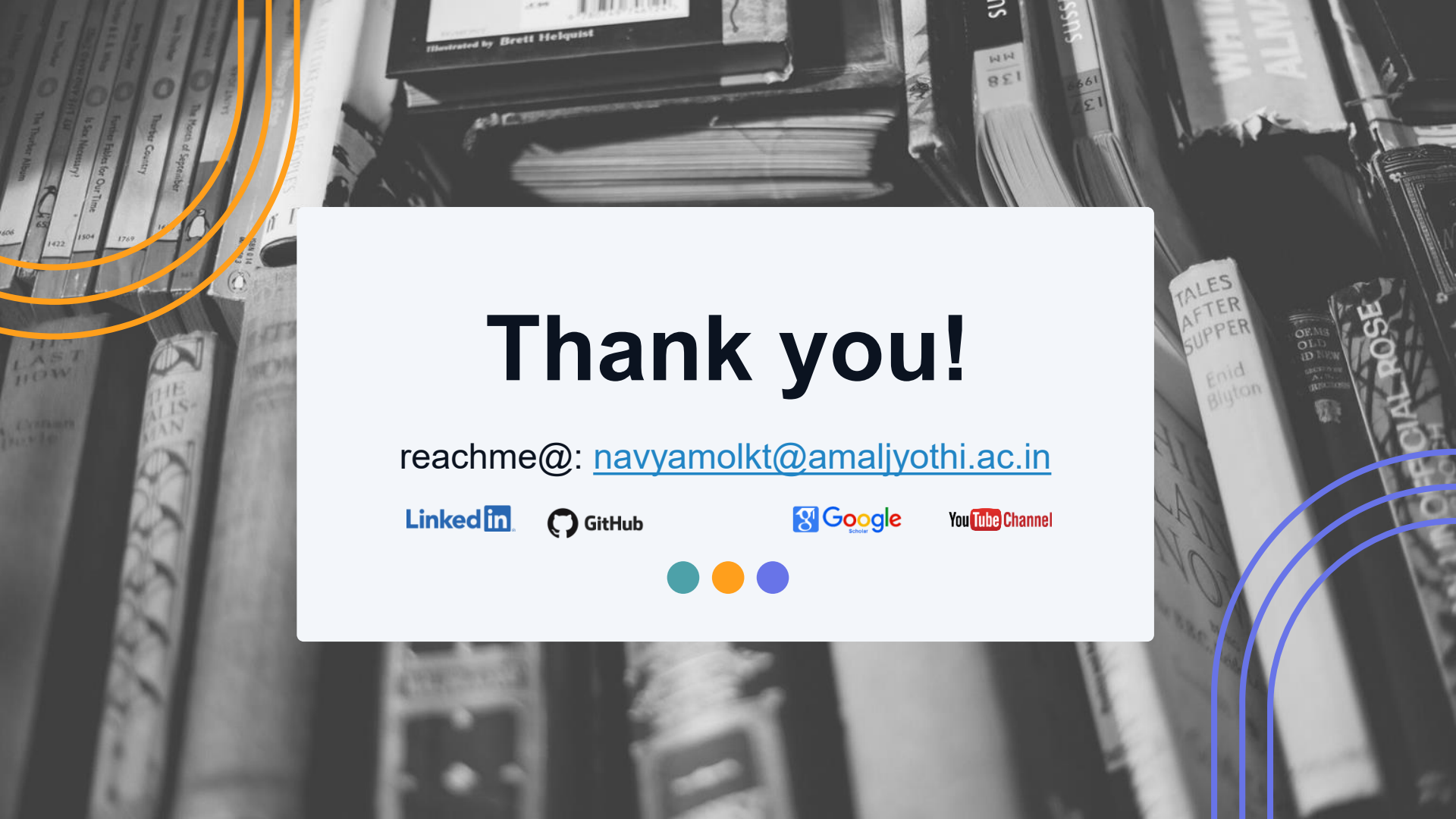
- Define each feasibility.
- Clarify how the feasibility is achieved in your project
- Please provide answers to the feasibility questions mentioned in the next slide.


T
Technology and System Feasibility
• Do stakeholders have the expertise needed?
• Are additional resources needed in the health system including infrastructure, skills-sets or job aids?
• Is the health system ready in terms of the technology required?
E
Economic Feasibility
• Do the resources needed exist?
• Will the proposed health service or initiative lead to better use of resources to improve health outcomes, when compared with other options?
L
Legal Feasibility
• Are rules and regulations in place to enable stakeholders to support the new service or initiative?
• Does the essential political will exist?
• Is there a legal framework to engage with the private sector or other key service providers?
O
Operational Feasibility
• Do existing health system procedures and protocols support the new service or intiative?
• How will key collaborators be involved?
S
Schedule Feasibility
• What are the prerequisites before the new service or initiative can begin?
• Is the service or initiative likely to be developed in time to be useful to the health system?

Useful APIs You can try out

Useful APIs You can try out

OpenWeatherMap API	• Provides current weather data and forecasts.
REST Countries API	• Offers information about countries (names, codes, population).
GIPHY API	• Supplies GIFs based on search queries.
JSONPlaceholder API	• Provides fake data for testing and prototyping.
IP Geolocation API	• Delivers geolocation data based on IP addresses
Quotes API	• Provides random or specific quotes.
JokeAPI	• Offers jokes of various categories and types.
Pexels API	• Supplies free stock photos and videos
NASA API	• Provides data related to NASA, including images and planetary information.
COVID-19 API	• Offers COVID-19 statistics and information globally or by country.



Thank you!

reachme@: navyamolkt@amaljyothi.ac.in

LinkedIn GitHub Google Scholar YouTube Channel