UML

# Objective:

To create the following UML:

* use-case diagram

for a real project or system.

# Theory:

**UML (UNIFIED MODELING LANGUAGE):**

**UML** (Unified Modeling Language) is a modeling language used by software developers. UML can be used to develop diagrams and provide users with ready-to-use, expressive modeling examples. Some UML tools generate program language code from UML. UML can be used for modeling a system independent of a platform language. UML is a graphical language for visualizing, specifying, constructing, and documenting information about software-intensive systems. UML gives a standard way to write a system model, covering conceptual ideas. With an understanding of modeling, the use and application of UML can make the software development process more efficient.

There are two categories of UML:

* behavior diagrams
* use case diagram
* sequence diagram
* structured diagrams
* class diagram
* **USECASE DIAGRAM:**

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

 A use case diagram contains four components.

The boundary, which defines the system of interest in relation to the world around it.

* The actors, usually individuals involved with the system defined according to their roles.
* The use cases, which are the specific roles played by the actors within and around the system.
* The relationships between and among the actors and the use cases.

**COMPONENTS in A Use Case Diagrams**

There are five types of relationships in a use case diagram. They are:

* Association between an actor and a use case
* Generalization of an actor
* Extend relationship between two use cases
* Include relationship between two use cases
* Generalization of a use case

**How to Create a Use-Case Diagram**

We are taking banking system as an example.

##### Identifying Actors

Actors are external entities that interact with your system. It can be a person, another system or an organization. In a banking system, the most obvious actor is the customer. Other actors can be bank employee or cashier depending on the role you’re trying to show in the use case.

An example of an external organization can be the tax authority or the central bank. The loan processor is a good example of an external system associated as an actor.

##### Identifying Use Cases

Now it’s time to identify the use cases. A good way to do this is to identify what the actors need from the system. In a banking system, a customer will need to open accounts, deposit and withdraw funds, request check books and similar functions. So all of these can be considered as use cases.

Top level use cases should always provide a complete function required by an actor. You can extend or include use cases depending on the complexity of the system.

Once you identify the actors and the top level use case you have a basic idea of the system. Now you can fine tune it and add extra layers of detail to it.

##### Look for Common Functionality to use Include

Look for common functionality that can be reused across the system. If you find two or more use cases that share common functionality you can extract the common functions and add it to a separate use case. Then you can connect it via the include relationship to show that it’s always called when the original use case is executed. ( see the diagram for an example ).

##### Is it Possible to Generalize Actors and Use Cases

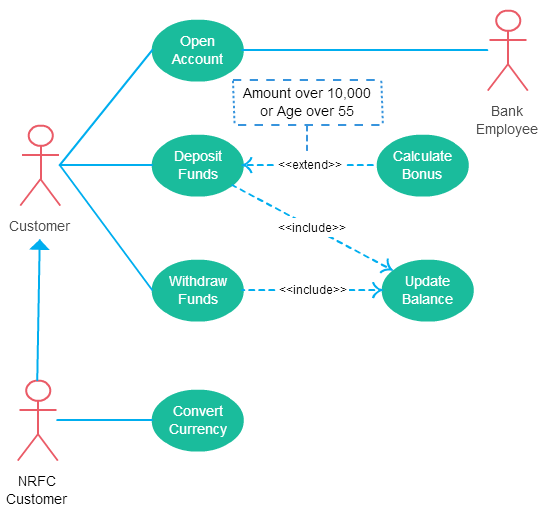
There may be instances where actors are associated with similar use cases while triggering few use cases unique only to them. In such instances, you can generalize the actor to show the inheritance of functions. You can do a similar thing for use case as well.

One of the best examples of this is “Make Payment” use case in a payment system. You can further generalize it to “Pay by Credit Card”, “Pay by Cash”, “Pay by Check” etc. All of them have the attributes and the functionality of a payment with special scenarios unique to them.

##### Optional Functions or Additional Functions

There are some functions that are triggered optionally. In such cases, you can use the extend relationship and attach an extension rule to it. In the below banking system example “Calculate Bonus” is optional and only triggers when a certain condition is matched.

Extend doesn’t always mean it’s optional. Sometimes the use case connected by extend can supplement the base use case. The thing to remember is that the base use case should be able to perform a function on its own even if the extending use case is not called.



**Exercise**

Draw use-case for the following scenario:

* Hotel management system

**Problem statement:**

A hotel has various types of rooms Dimensions of various: price, number of single beds, number of double beds. A database with a listing of all the rooms of the hotel is supplied. This database includes when the rooms have been booked.

People can look for availability on a website for certain types of room (room price), for a certain time of span. The systems check availability and returns the proposition that fits the reservation. If no exact match is found, something similar is proposed with at least the same person capacity.

* Airline reservation system

**Problem Statement:**

To develop a computerized meeting, the rising customer interest in booking online air travel reservations. The system should be convenient, user friendly and available via the internet.

The system should allow the users to view entire flights information of the airline, book tickets, view or, if required cancel current reservations and create member login for standalone users as well as agents.