**PART A**

**EXPERIMENT NO. 2**

**A.1 Aim: - Modeling UML Use Case diagrams and capturing Use Case scenarios.**

**A.2 Prerequisite**

**1. Concepts of Actor, Use Case and Relationships**

**A.3 Outcome**

After successful completion of this experiment students will be able to

Design solution using unified modeling language.

**A.4 Theory**

**Use case diagrams**

Use case diagrams belong to the category of behavioural diagram of UML diagrams. Use case diagrams aim to present a graphical overview of the functionality provided by the system. It consists of a set of actions (referred to as use cases) that the concerned system can perform, one or more actors, and dependencies among them.

**Actor**

An actor can be defined as an object or set of objects, external to the system, which interacts with the system to get some meaningful work done. Actors could be human, devices, or even other systems.

For example, consider the case where a customer *withdraws cash* from an ATM. Here, customer is a human actor.

Actors can be classified as below :

* **Primary actor**: They are principal users of the system, who fulfill their goal by availing some service from the system. For example, a customer uses an ATM to withdraw cash when he needs it. A customer is the primary actor here.
* **Supporting actor**: They render some kind of service to the system. "Bank representatives", who replenishes the stock of cash, is such an example. It may be noted that replenishing stock of cash in an ATM is not the prime functionality of an ATM.

In a use case diagram primary actors are usually drawn on the top left side of the diagram.

**Use Case**

A use case is simply a functionality provided by a system. Continuing with the example of the ATM, *withdraw cash* is a functionality that the ATM provides. Therefore, this is a use case. Other possible use cases includes, *check balance*, *change PIN*, and so on.

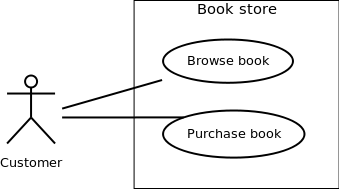
Use cases include both successful and unsuccessful scenarios of user interactions with the system. For example, authentication of a customer by the ATM would fail if he enters wrong PIN. In such case, an error message is displayed on the screen of the ATM.

**Subject**

Subject is simply the system under consideration. Use cases apply to a subject. For example, an ATM is a subject, having multiple use cases, and multiple actors interact with it. However, one should be careful of external systems interacting with the subject as actors.

**Graphical Representation**

An actor is represented by a stick figure and name of the actor is written below it. A use case is depicted by an ellipse and name of the use case is written inside it. The subject is shown by drawing a rectangle. Label for the system could be put inside it. Use cases are drawn inside the rectangle, and actors are drawn outside the rectangle, as shown in below figure:



Association between Actors and Use Cases

A use case is triggered by an actor. Actors and use cases are connected through binary associations indicating that the two communicates through message passing.

An actor must be associated with at least one use case. Similarly, a given use case must be associated with at least one actor. Association among the actors are usually not shown. However, one can depict the class hierarchy among actors.

**Use Case Relationships**

Three types of relationships exist among use cases:

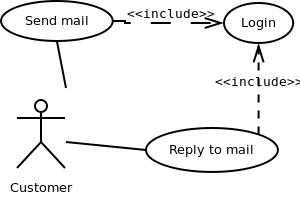
* Include relationship
* Extend relationship
* Use case generalization

**Include Relationship**

Include relationships are used to depict common behaviour that are shared by multiple use cases. This could be considered analogous to writing functions in a program in order to avoid repetition of writing the same code. Such a function would be called from different points within the program.

**Example**

For example, consider an email application. A user can send a new mail, reply to an email he has received, or forward an email. However, in each of these three cases, the user must be logged in to perform those actions. Thus, we could have a *login* use case, which is included by*compose mail*, *reply*, and *forward email* use cases. The relationship is shown in below figure.



**Extend Relationship**

Use case extensions are used used to depict any variation to an existing use case. They are used to the specify the changes required when any assumption made by the existing use case becomes false.

**Example**

Let's consider an online bookstore. The system allows an authenticated user to buy selected book(s). While the order is being placed, the system also allows to specify any special shipping instructions, for example, call the customer before delivery. This *Shipping Instructions* step is optional, and not a part of the main *Place Order* use case. Below figure depicts such relationship.

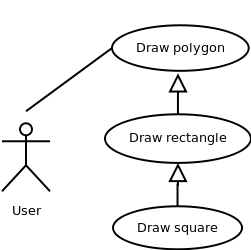
<<extends>>

**Generalization Relationship**

Generalization relationship are used to represent the inheritance between use cases. A derived use case specializes some functionality it has already inherited from the base use case.

**Example**

To illustrate this, consider a graphical application that allows users to draw polygons. We could have a use case *draw polygon*. Now, rectangle is a particular instance of polygon having four sides at right angles to each other. So, the use case *draw rectangle* inherits the properties of the use case *draw polygon* and overrides it's drawing method. This is an example of generalization relationship. Similarly, a generalization relationship exists between *draw rectangle* and *draw square* use cases.



**A.5 Task:**

For the problem statement, complete use case modelling in StarUML.

**PART B**

|  |  |
| --- | --- |
| Roll No: B017 | Name: Niharika Dalal |
| Class: B. Tech. Comp. Engg. | Batch: B1 |
| Date of Experiment: 11.1.2015 | Date of Submission: 18.1.2015 |
| Grade: A+ |  |

**B.1 Actors:**

**………………………………………………………………………………………**

Actor 1: Customer: who orders from the app

Actor 2: Server: who controls the back-end and the delivery of products.

Actor 3: Public (friends): The user can share/ publicize the app.

**………………………………………………………………………………………**

**B.2 Use cases:**

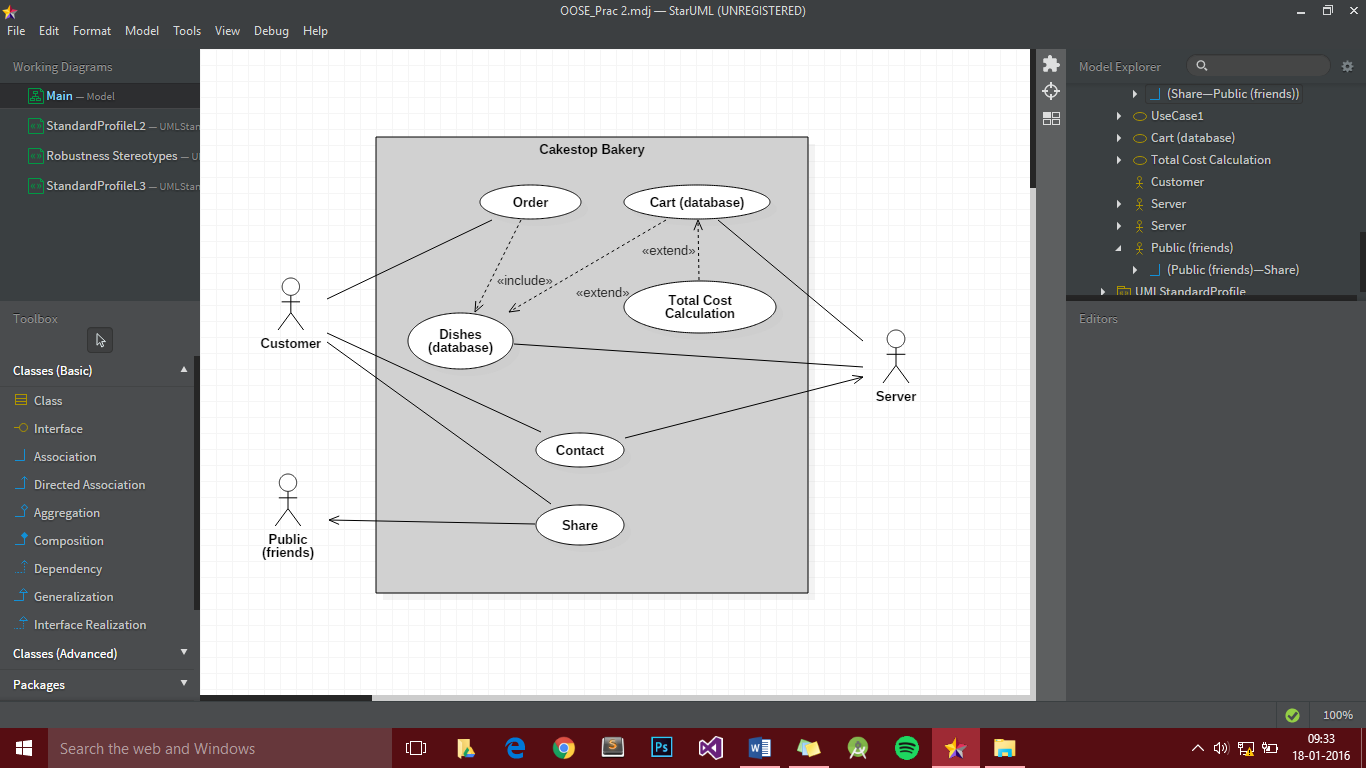
**………………………………………………………………………………………**

1. Order: Ordering of products
2. Dishes (database): Contains all the dishes available at the bakery.
3. Cart (database): When the customer selects a product and wishes to buy it, he ads it to the cart. More than one products can be added to the cart and the total amount is displayed.
4. Total Cost Calculation: Calculates the total cost of the products in the cart.
5. Share: The app can be shared by the user/server to his/her friends.
6. Contact: The users can contact the bakery in case of queries.

**………………………………………………………………………………………**

**B.3 Use Case diagrams:**

**………………………………………………………………………………………**



**………………………………………………………………………………………**

**B.4 Use Case Specifications:**

**………………………………………………………………………………………**

Customers can order bakery products available at Cakestop. This is an ‘**include**’ relationship because when the user opens the app, he is shown the dishes from which he can order. The products available (dishes) are fetched from the dishes database and displayed. The user, if he wants to order, can add one product at a time into the cart. This is an ‘**exclude**’ relationship because it is not necessary that the user will add items to the cart. The total cost of all the products is calculated and displayed. The server manages the ordering of items from the cart, along with the total cost. In case of any query, the customers/users can contact the bakery, by either calling or using the short-message-service (SMS). The user can also share/publicize the app by sharing it.

**………………………………………………………………………………………**

**B.5 Conclusion**

**………………………………………………………………………………………**

A **use case** is a list of actions or event steps, typically defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system, to achieve a goal. The **actor** can be a human, an external system, or time.

It also shows the relationships (dependencies) between two use cases, like ‘include’, ‘extend’ and ‘generalization’.

**………………………………………………………………………………………**

**B.5 Questions of Curiosity:**

Q1. What does a use case diagram represent?

1. A set of actions
2. Time sequence of statements executed
3. How to use a particular module
4. Don’t know

Answer: **a. A set of actions**

Q.2 Generalization relationship exists between two use cases when

1. A use case derives from a base use case.
2. A use case derives from a base use case and specializes some of its inherited functionality.
3. A use case include functionality of some other use case.
4. No two use cases can be related.

Answer: **b. A use case derives from a base use case and specializes some of its inherited functionality.**