COE528 Fall 2018 Final Project

Name: Ayush Dave Student ID: --------- Section:09

Use Case Diagram Description:

The use case for the project contained two actors (Customer and Manager). It also contained the following 8 use cases: Login, Logout, Deposit, Withdraw, Online Purchase, Balance, Delete Customer and Add Customer. There was a "Communications" relationship between the customer actor and the Login, Logout, Deposit, Withdraw, Online Purchase and Balance use cases, while there was a "Communications" relationship between the manager and the Login, Logout, Delete Customer and Add Customer use cases. This relationship was depicted by a solid line between the actors and the use cases. There was a "extends" relationship between the Login use case and all the other use cases since, the actors do not always have to be involved every use case once logged into the system. This relationship was depicted by dashed lines with the word "extends" originating from each use case to login. There was an "Includes" relationship between login use case and logout use case since the actor always has to logout after logging in. This relationship was depicted by dashed lines with the word "includes" originating from the including login use case to logout.

"Withdraw" Use Case Description:

Use Case Name: Withdraw

Participating Actors: Customer

Flow of Events:

1. The customer inputs the amount of money they would like to withdraw in the text field
2. The customer clicks the "Withdraw" button
3. The system prints out a "Successful Withdrawal" or error message depending on the amount of money to withdraw inputted by the customer

Entry Condition:

1. The customer is logged into the system
2. The customer clicks the "withdraw" button in the customer control menu

Exit Conditions:

1. The customer has successfully made a withdrawal with a "Successful Withdrawal" OR
2. The customer is unsuccessful in making a withdrawal OR
3. The customer clicks the "Back" button

Exceptions:

The system prints out an error message if the withdrawal amount is negative, zero or greater than the current balance of the customer.

Special Requirements: The withdrawal amount entered must be greater than zero and less than the current balance.

Class Diagram Description:

The class diagram consisted of the following 5 classes: Customer, State, Silver, Gold and Platinum. The customer class contained instance variables that held the information of the customer such as what level customer they were (Silver, Gold or Platinum), the current balance, deposit/withdraw amount, online purchase amount, username and password of their account. The methods of the class allow for altering the instance variables such as when the customer level changes or a deposit/withdrawal/online purchase was made. It also allowed the instance variables to be visible to the user of the system through getter methods. The state class was an abstract class which had an aggregate (has-a) relationship with the Customer class as well as an inheritance (is-a) relationship with its children Silver, Gold and Platinum. It contained a single abstract method called changeState taking in a customer object as a parameter, which was overridden by its children methods Silver, Gold and Platinum. The Silver, Gold and Platinum classes each overrode the changeState method from the State class by altering the state of the customer object to Silver, Gold or Platinum depending on if the balance was less than $10000, greater than $10000 but less than $20000, and greater than $20000. These three classes also had toString() methods returning their class names as a string.

State Pattern Classes and Documentation of Customer Class

The class addressed in question two was the Customer class. The REQUIRES, MODIFIES and EFFECTS clauses were written for each method. The OVERVIEW clause was written as well as the abstraction function and the rep invariant. The rep invariant was implemented in the repOk() function and the abstraction function was implemented in the toString() function. The Customer, State, Silver, Gold and Platinum classes make up the state pattern as the customer class represents the Context class which sends a state-specific request to the concrete State subclasses which are Silver, Gold and Platinum. These subclasses decide which concrete state subclass the current customer object will be set to depending on the balance of the customer object.