Muhammad Rohan Zahid – 500896085 – Section 7

Simple Banking Application Report

The UML Use Case Diagram for this application describes the functional behaviour of the system as perceived by the user. This Banking client is designed to be used by a Manager and Customers. The Manager can accomplish four tasks with this client: login, logout, add a Customer, and delete a Customer. In order to add a Customer or delete a Customer, the Manager must be logged in. If the correct credentials are not provided at login (username: admin, password: admin), the Manager will not be logged in. If the Manager enters the username of an existing Customer when adding a new Customer, a new Customer will not be added. Likewise, if the account balance of the new Customer is less than $100, a new Customer will not be added. If any invalid inputs are given when adding a new Customer, such as spaces between the username and/or password or a non-numeric account balance, a new Customer will not be added. Furthermore, if there are zero customers at the time, the Manager cannot remove anyone. If the Manager successfully logs in, he/she will be able to logout. Therefore, the logout, add Customer, and delete Customer cases extend from login. The Manager can choose to do those tasks or just exit the program. A Customer can accomplish six tasks: login, logout, deposit money, withdraw money, get balance, and online purchase(s). If the correct credentials are not provided at login (a matching username and password from the pre-existing Customer files), the Customer will not be logged in. In order to deposit money, withdraw money, get balance, make online purchase(s), or logout, the Customer must be logged in. If the Customer enters a non-numeric value or a numeric amount less than $1, the deposit will not be made. The same conditions apply when making a withdrawal, but the withdrawal will also not be made if the Customer's bank balance is less than the amount to be withdrawn. If the Customer successfully logs in, they will be able to check their bank balance or logout without restriction. Similarly, if the Customer enters a non-numeric value, a numeric amount less than $50, or a numeric amount that exceeds the Customer's current bank balance with the added fee, the online purchase will not be made. Therefore, the logout, deposit money, withdraw money, get balance, and online purchase(s) cases extend from login.

The UML Class Diagram for this application consists of various classes, which interact with each other in different ways. The abstract *User* class contains three Strings (*username*, *password*, and *role*), three methods (*getUsername()*, *getPassword()*, and *getRole()*), and a Constructor. The concrete *Manager* and *Customer* classes inherit from the *User* class. The *Manager* class has an integer called *currentCustomer*, which holds the index of the *Customer* that is currently logged in. The *Manager* class also has an ArrayList of *Customer* objects called *customers*. There is only one *Manager* and zero or more *Customers*. The *addCustomer(Customer c)* method saves a *Customer's* information to its corresponding text file and adds the *Customer* to the ArrayList *customers*. The *removeCustomer(String s, String x)* method deletes a *Customer's* text file and removes the *Customer* from the ArrayList *customers*. The *setCurrentCustomer(int i)* method sets the value of *currentCustomer* to the integer value given as a parameter. The *getCurrentCustomer()* method returns the value of *currentCustomer* to the invoker. The *doesUserExist(String username)* method returns true if the username, given as a parameter, is already in the ArrayList *customers*; returns false, otherwise. The *login(String username, String password)* method compares the credentials passed by the user to the pre-set credentials of the Manager. If the credentials match, the method returns true; returns false, otherwise. The *Customer* class has a *Level* object called *level*, which is instantiated as a *Silver*. The

setLevel(Level l) method sets the value of *level* to the *Level* value given as a parameter. The *getLevel()* method returns the value of *level* to the invoker. The *checkLevel()* method gives the responsibility of changing the state of *level* to the concrete *Level* classes. The *login(String username)* method checks if the username entered by the user is the same as the username saved in the *Customer* account. If it is the same, the method returns true; returns false, otherwise. The *repOK()* method returns true if the rep invariant holds for the *Customer* object; otherwise returns false. The *toString()* method returns a string that contains the identifying information about the current Customer and implements the abstraction function. Furthermore, the *Customer* class is an aggregate of the *BankAccount* class and the *Level* class. Each *Customer* has one *BankAccount* and each *Customer* has one *Level*; one-to-one. The *BankAccount* class has a double called *amount*, which holds the amount of money a *Customer* has in their account. The *getAmount()* method returns the value of *amount* to the invoker. The *setAmount(double amount)* method sets the value of *amount* to the *double* value given as a parameter. The abstract *Level* class has a *double* called fee, an abstract *checkLevel(Customer c)* method, and an abstract *getFee()* method. The concrete *Silver*, *Gold*, and *Platinum* classes inherit from *Level*. They all override the abstract methods from *Level*. The *checkLevel(Customer c)* method checks the amount of money the *Customer* has in their *BankAccount* and changes the state of *Level* accordingly. The *getFee()* method returns the value of *fee* for the corresponding state of *Level*.

The class selected for point number 2 from the *Work Items* is *Customer.java*.

From the UML class diagram, the *Customer*, *Level*, *Silver*, *Gold*, and *Platinum* classes form the State design pattern. The *Customer* class is the context, as it delegates the state-specific request to the current concrete “state” object, which in this case is the current concrete *Level* object. Since the change of *Level* of the *Customer* is the responsibility of the concrete state classes, the state transition logic is incorporated into the state objects. For instance, if the *Customer* has an amount of $20000 or more in their *BankAccount*, they have a *Level* of *Platinum*. If the *Customer* has an amount of less than $20000 but at least $10000 in their *BankAccount*, they have a *Level* of *Gold*. If the *Customer* has an amount less than $10000 in their *BankAccount*, they have a *Level* of *Silver*.