Name: Shaan Hossain
Student Id#: 500882569
Section#: 2

Use Case Diagram Description

My Use Case diagram initially starts with an arbitrary user that interacts with the Login use case and must authenticate with an appropriate username and password (includes relationship), in order to be set as a Manager user or a Customer user. If the user is authenticated as a Manager, the user is now able to access the use cases of Add Customer, Log Out and Delete Customer where the Add Customer use case creates a file (extends relationship) then uses external functions of reading both username and password entered by the user(includes relationship) to successfully add a customer and write their information to the file. On the other hand, if the user accesses the Delete Customer use case, external functions are used to read the username of the customer to be deleted (includes relationship) then delete the corresponding file (extends relationship). If the user is authenticated as a Customer, the user is now able to access the use cases of Show Balance, Do Online Purchase(s), Log Out, Withdraw, and Deposit. Then, if the user accesses the use cases of Do Online Purchase(s), Withdraw, or Deposit, then each uses an external function to read a value (includes relationship), respectively, then validate the corresponding value (hence extends relationship) to perform their own respective functions. However, unlike the other use cases accessed by a Customer user, the Do Online Purchase(s) must also perform a fee calculation in addition to getting a value. (extends relationship) Lastly, whether the user is logged as a Manager or Customer, both access the Log Out use case to leave their respective interface and return to the Login use case as an arbitrary user.
(extends relationship)

UML Class Diagram Description

My UML Class diagram composes of 8 different classes that all implement different functions to perform the required tasks of this application. Firstly, I have the Customer class, which has key instance variables role, username, and password to define a unique Customer object. This class is also composed of two other key classes, Level and Account, in a one-to-one multiplicity, since the specification requires that each customer must have exactly one Account and one varying Level. In more detail, the Level class is an abstract class that is a parent of an inheritance hierarchy where the Level object varies (State Design) depending on its instantiation to one of its 3 children classes: Silver, Gold, Platinum. Each child implements their own methods of the onlinePurchase function and as such, will have varying online purchasing fees. As for the Account Class, since it is composing the Customer class in a 1-1 multiplicity, can be delegated responsibility of the Customer class and as such, the withdraw, deposit, and getBalance functions are implemented into the Account class. Secondly, for the Manager Class, key instance variables role, username, and password are defined as “Manager”, “admin”, and “admin”, respectively, to indicate that a Manager can only be defined with “admin” for both username and password. Moreover, this class is responsible for creating, deleting, and managing Customers and their

Name: Shaan Hossain
Student Id#: 500882569
Section#: 2

corresponding files by keeping a list of Customers (0 to many aggregation relationship) and performing various of file input and output practices (FileWriter Class, Scanner Class, File, Path Directory, etc) for changes in these files. Lastly, for the UserInterface Class, it depends on both the Manager and Customer Classes (dependency relationship) since information from both is used to authenticate and give access to their corresponding interfaces. As a result, instance variables such as Customer ctemp and Manager M are declared to provide a structure centered around these two classes. As for the functions of this class, it is mainly for implementing the GUI of this application with JavaFX.

Addressing Point 2

The class that wrote and implemented the Abstraction Function, Rep Invariant, Overview Clause and the Effects Clause in this application is the Customer Class.

Formation of the State Design Pattern

As mentioned in the UML Class Diagram Description, the State design pattern is implemented in the inheritance hierarchy between the Level Abstract Class and its children classes: Silver, Gold, and Platinum. Also, it is implemented through the fact that all the concrete class(children) are responsible for the change of the Level object through the inherited updateLevel function and as such, completes the implementation of the State design pattern.

References

- JavaFX documentation : https://docs.oracle.com/javase/8/javafx/api/toc.htm
- Learning JavaFX : http://tutorials.jenkov.com/javafx/index.html