UML Use Case Diagram

The Use Case Diagram represent the external behaviour of the Bank Account Application. The actors consists of a Customer and Manager user that both interact with the system. Both actors have communication relationships with distinct use cases but one – UseBankAccount use case. Users communicate with this use case as the entry point for the system which includes a Logout and Login use case and in particular, the Login use case has multiple extends relationship to handle exceptions. When the login attempt is successful, a Manager is then able to access the AddCustomer and DeleteCustomer use cases. The AddCustomer use case has two extends relationship – one to handle invalid input such as a null input and the second to handle if a customer already exists. The DeleteCustomer use case only has one extends relationship which is to cancel the deletion of a customer. Likewise, when the login attempt is successful, a Customer is then able to access the CheckBalance, DepositMoney, WithdrawMoney and DoOnlinePurchase use cases. DoOnlinePurchase, WithdrawMoney and DepositMoney all extends an InvalidInput use case which handles invalid input as in the name. The WithdrawMoney and DoOnlinePurchase includes a CheckFunds use case which extends multiple two use cases to handle exceptions such as insufficient funds and to cancel the action. The DoOnlinePurchase use case has an extends relationship with the LimitNotReached use case which is to handle when the purchase amount is under the limit threshold. In addition, DoOnlinePurchase has three inheritance relationships that will specialize the use case based on the level of the Customer. Finally, the exit point of the system is when the Users communicate with the Logout use case.

UML Class Diagram

The Class Diagram represent the structure of the Bank Account Application. The client class serves as the main interaction between the user and the system. This class also employs the Graphical User Interface (GUI) aspect (i.e. the pages) of the Bank Account Application. The Users class is an abstract class that is the parent of the concrete Manager and Customer sub-classes. It provides the general structure as well as the specific implementations for each sub-class. It has overloaded constructors in addition to protected and normal methods that will be overridden and implemented in each corresponding sub-class. The Client and User class has an aggregation relationship with one another. The Client owns the User class where the Client must have one or more Users and conversely the Users must only have one instance of the Client class. The Manager class is a concrete sub-class of the Users class and manages the implementation and methods corresponding to when a User is of a type manager. Similarly, the Customer class is a concrete sub-class that manages the implementation and methods corresponding to when a User is of a type customer. Note that the Manager and Customer class has implemented a ‘dummy’ singleton pattern in order to instantiate the Users type when the

Client calls the handleLogin method. The Customer class has a composite relationship with the Bank Account class. There is one instance of Bank Account for an instance of Customer and vice versa. Finally, it is important to note that all of the classes are mutable except the Manager class. This is because the state of an object has to change during run-time to display appropriate changes according to the requirements of the application.

Abstraction

For the example of the abstraction of the Bank Account Application, please refer to the Bank Account class. Note that the Javadoc file is located at:

\project\dist\javadoc\project\BankAccount.html

Design Pattern

From the UML class diagram, the Client, Users, Manager and Customer classes form the State Design Pattern. The Client is the primary interface for the user. The Client is responsible for the responsiveness of the GUI of the application by delegating state-specific requests to the concrete state objects (i.e. Manager and Customer). The concrete sub-classes will handle the logic of the requests such that it localizes the behavior of the state specific response associated with the current state of the user. The Client class has a Users instance variable that is instantiated by the concrete sub-classes. For example when the user has to login, the Client will request a login attempt according to the state the user picked. If the user is a manager, the Manager class will handle the request and similarly if the user is a customer, the Customer class will handle the request.