Final Project: Bank Account Application

Petar Zmak

501188283

Section 08

COE528: Object Oriented Eng Analysis and Design

Boujemaa Guermazi

March 23rd, 2024

Petar Zmak
501188283
Section 08

Use Case Diagram

This Use Case Diagram describes the behavior of the Bank Account Application. There are two actors in the diagram: Manager and Customer. Both actors can use the application, the actors will have access to different use cases depending on if the actor is a Manager or a Customer. If the actor is a Customer, they will have access to five different use cases. A Customer can deposit money, withdraw money, make an online purchase, check their balance, or log out. These actions are represented as five use cases colored gray in the diagram. To do any of the five actions, the Customer must first log in. This is shown in the diagram by the "include" relationship between the "Log in" use case and each of the other five use cases. The Customer also is able to cancel certain actions. If the Customer is withdrawing money, depositing money, or making an online purchase, the Customer can cancel the transaction at any time. This is shown by the "extends" relationship between the use case and the "Cancel" use case. If the actor is a Manager, they will have access to three different use cases. A Manager can create a Customer, delete a Customer, and log out. These actions are represented as three use cases colored gray in the diagram. To perform any of the three actions, the Manager must first log in. This is shown in the diagram by the "includes" relationship between each use case and the "Log In" use case. The Manager is able to also cancel certain actions. If the Manager is creating or deleting a Customer, they can cancel the action. This is shown by the "extends" relationship between the use case and the "Cancel" use case.

Use case name	`Withdraw Money`
Participating Actors	Initiated by `Customer`
Flow of Events	1. The `Customer` logs in with their username and password 2. The `Customer` selects the "Withdraw" option on their home page 3. The `Customer` enters the amount of money they wish to withdraw, not more than the `Customer` current balance 4. The `Customer` clicks the "withdraw" button 5. The application displays the amount withdrawn
Entry Condition	● The `Customer` is logged into their account ● The `Customer` does not enter more money than is in their balance
Exit Condition	● The `Customer` has received their money ● The `Customer` has received an error message
Quality Requirements	● None

Petar Zmak
501188283
Section 08

Class Diagram

The Class Diagram describes the structure of the Bank Application. The main method is located in the App class. There are eight JavaFX Controller classes in the application, each which controls a different page in the application. The eight controller classes are: LogInPageController, ManagerHomeController, CustomerHomeController, WithdrawController, DepositController, OnlinePurchController, AddCustomerController, and DelCustomerController. Each of these classes use the setRoot method in the App class. This is shown in the diagram by the dependency relationship between each of the Controller classes and the App class. The LogInController creates an AppStart object which is shown by the aggregation relationship between the LogInController class and the AppStart class. The AppStart class verifies the username and password taken in by the LogInController, and then creates a new User which is shown by the aggregation relationship between the AppStart class and the User class. The User class is an abstract class and it has two children: Customer and Manager, which is shown by the inheritance relationship between the User class and the Manager and Customer class. When the AppStart class creates a new User, based on the information in the file it reads from, it will create a new Customer or a new Manager. The abstract User class will also create a new AccountLevel object. This is shown by the aggregation relationship between the AccountLevel class and the User class. The AccountLevel class has three children: Silver, Gold, and Platinum which is shown by the inheritance relationship between the AccountLevel class and the three subclasses. Depending on the level of the Customer, the User class will create the AccountLevel object as a new Silver, Gold, or Platinum object. When a new Customer object or Manager object is created, they create a new BankAccount object. This is represented by the aggregation relationship between the Customer or Manager class and the BankAccount class. The BankAccount class has methods for all of the functions the Customer and Manager can do (all the methods for the functions are in the BankAccount class, but certain methods are only called depending on if the user is a Customer or a Manager). Each of the controllers specified above, also access certain methods in the User or Customer class to get user of customer specific information. This is shown by the dependency relationship between all the controller classes except LogInPageController, and the User class. Four controller classes: CustomerHomeController, WithdrawController, DepositController, and OnlinePurchController, also access methods in the Customer class to get Customer specific information. This is shown by the dependency relationship between each of the specified four classes and the Customer class.

The parts of the Class diagram that form the State Design pattern are: the abstract User class, the abstract AccountLevel class, the Silver class, the Gold class, and the Platinum class. In this State pattern the User class is the Context, the AccountLevel is State, and the Silver, Gold, and Platinum classes are the ConcreteState subclasses.