Project Report

Simple Banking Application

By:
Mariam Ahmad

COE 528
March 25, 2024

Introduction

In this project, a graphical user interface in JavaFX was used to create a simple bank application that would serve the customers and manager with standard banking functions. This report includes two UML representation diagrams, implementation of the project's State design pattern and an expanded definition of the Customer class.

Use Case Diagram

The Use case diagram created to represent the bank application is depicted in Figure 1.

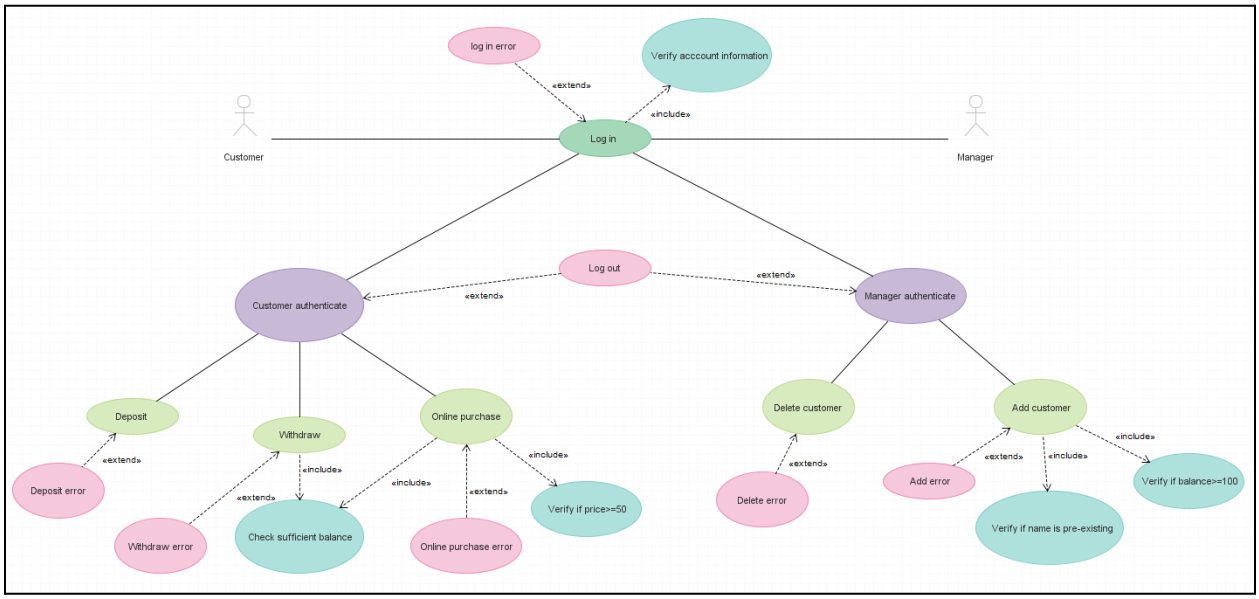

Figure 1: UML Use case diagram

The purpose of this Use case diagram is to showcase the functional behavior of the bank application and how the actors will interact with it. The two primary actors in this application are the customer and manager. The use cases represent specific tasks that can be performed by the actors using the application. The manager once logged in will have access to the functions; "Add customer," "Delete customer" and "Log out." The customer once logged in will have access to the functions; "Deposit," "Withdraw," "Online purchase" and "Log out."

The Use case can be represented textually, displayed in Table 1.

Use case name	Add customer
Participating actors	Initiated by Customer, communicates to Manager
Flow of events	1. Manager will log into the bank application 2. Manager will ask customer for their username, password and balance

	3. Customer will provide the login parameters and give a balance of $200 4. Manager will input the information and add customer 5. Manager will then logout of the bank application
Entry condition	Manager is logged into the bank application
Exit condition	Adding the customer is successful and the Manager logs out
Exceptions	There are no exceptions for this scenario
Special requirements	The balance must be greater than or equal to $100 and the username cannot be the same as a pre-existing customer

Table 1: Textual Use case representation

Class Diagram

The Class diagram created to represent the bank application is depicted in Figure 2.

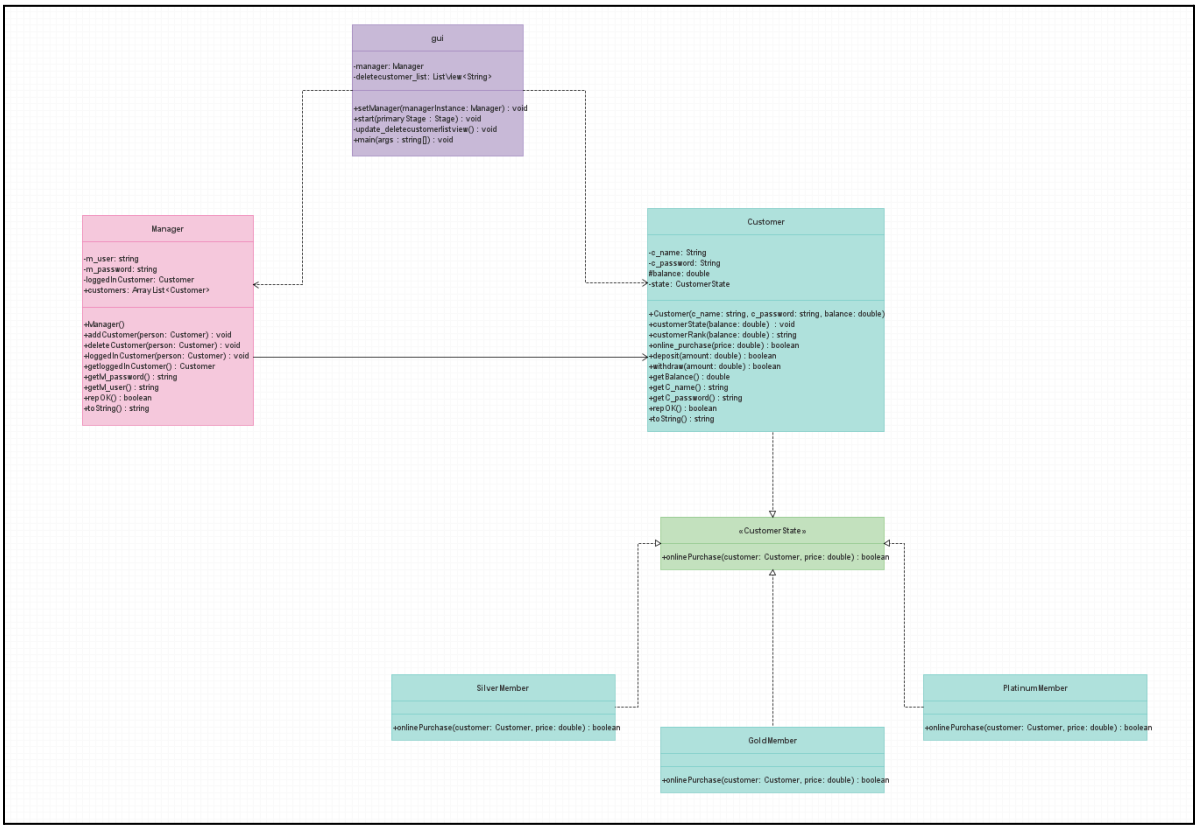

Figure 2: UML Class diagram

The purpose of this Class diagram is to showcase the static structure of the banking application which contains the objects, attributes and associations. This diagram consists of three main classes; Customer, Manager and gui. The Manager and Customer class contain functionalities that are specific to their roles, whereas the gui class serves as the graphical interface in which the users will interact with the application.

State Design Pattern

A State design pattern is to allow the change of behavior for an object without the change of class. In the bank application all customers, depending on their balance, are divided into three ranks, Silver Member, Gold Member or Platinum Member. The program will implement a State design pattern by defining three concrete classes for each rank. These concrete classes will implement the CustomerState interface which will define a method for online purchases that is dependent on the state/rank of the customer. The Customer class will also implement the CustomerState interface utilizing the methods; ‘customerState’ and ‘online_purchase.’ Depending on the current state of the customer, when ‘online_purchase’ is called, it will call the ‘onlinePurchase’ method in its respective concrete class, this allows it to showcase the State design pattern. These relationships can be seen in Figure 2.

Customer Class

The Customer class was selected to implement the abstraction function, representation invariant and all other required clauses. This class will initialize a Customer object that contains the login information, balance and state. The abstraction function is used to demonstrate how the instance variables link to the object they should represent, which in this application is a customer. The method ‘toString’ is used to implement the abstraction function. The purpose of the representation invariant is to verify the assumptions made by the methods in the Customer class. This is implemented by the method ‘repOk,’ which will return true if assumptions are correct.

References

Toronto Metropolitan University. 2024. COE528 (Winter 2024) - Project.