Toronto Metropolitan University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Object Oriented Eng Analysis and Design
Course Number:	COE528
Semester/Year (e.g. F2016)	W2024

Instructor:	Boujemma Guermazi

Assignment/Lab Title:	COE528 Term Project

Submission Date:	March 25, 2024
Due Date:	March 25, 2024

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Sin	Darren	501167165	12	

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:

https://www.torontomu.ca/senate/policies/pol60.pdf

COE528 - FINAL PROJECT REPORT

INTRODUCTION

The goal of this project is to develop a simple banking application through the use of the Java programming language. The application will include a graphical user interface built using JavaFX. This report will outline the design, structure, and implementation of the system.

PROJECT OVERVIEW

The banking application allows users to interact with a simple banking system in two ways depending on their role. These roles consist of a Manager and a Customer. The application will allow customers to login, logout, perform transactions (i.e. deposits, withdrawals, and online purchases), and get balance. Depending on the customer's level, a fee will be applied to online purchases. A manager can login, logout, add customers, and delete customers. This application employs algorithms, file handling, object-oriented programming strategies, data structures, and an attractive GUI to ensure an enjoyable user experience.

USE CASE DIAGRAM

Figure 1. The Use Case Diagram of the Banking Application

The use case diagram illustrates the interaction between the users of the application and the system. As shown from the user case diagram of **Figure 1**, there are different actions and interactions with the system depending on the type of actor–Manager and Customer. The main actions a manager performs are login, logout, add customer, and delete customer. The main actions a customer performs are login, logout, make deposit, make withdrawal, make online purchase., and view balance. The use case that will be analyzed is the make withdrawal use case. The participating actor is a customer. The use case begins when the customer successfully logs in. Next, the customer will be able to make a deposit by selecting the withdrawal type transaction, entering a numerical value in the text field, and pressing the submit button. The system will check whether the value entered is positive and if the customer has the sufficient funds to make a withdrawal. If the checks are valid, it will take money out of the account and return a success message. Otherwise, it will display an error message. After the message, the use case will terminate.

CLASS DIAGRAM

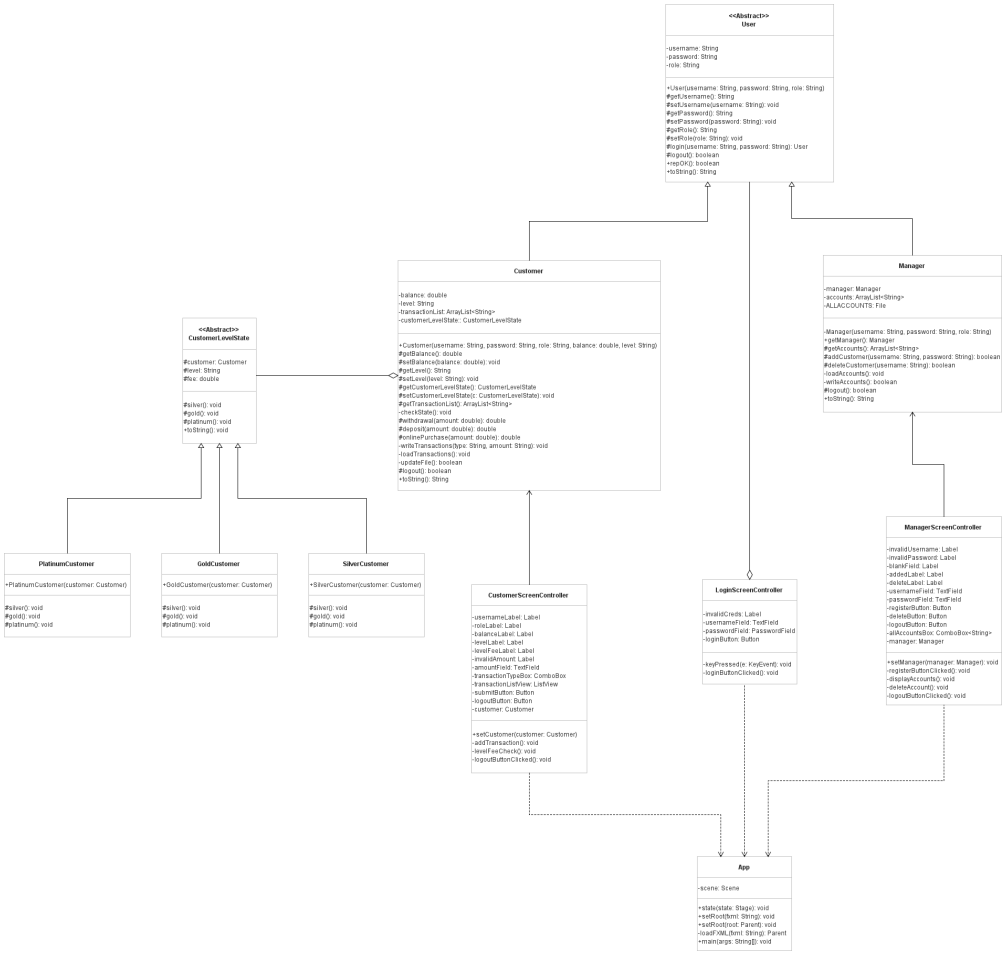

Figure 2. The UML Class Diagram of the Banking Application

The diagram above depicts the UML class diagram of the banking application. The User, Manager, and Customer classes define the type of users of the application. Since the Manager and Customer extend the abstract User class, the arrows depict an “is-a” relationship. CustomerLevelState is an abstract class that defines characteristics of the SilverCustomer, GoldCustomer, and PlatinumCustomer concrete classes. This is depicted by an arrow that shows a “is-a” relationship. Since the Customer class contains a CustomerLevelState, it exerts a “has-a” relationship arrow with the class. The LoginScreenController contains a user before logging in and thus shows a “has-a” relationship with the User class. The other screen controllers are associated with either a Customer or Manager object and represent the current user using the application. Hence, a “has-a” relationship arrow is established between the CustomerScreenController and ManagerScreenController class, and the Customer and Manager object class respectively. The last class presented on the diagram is the App class. The App class is the class that controls all of the GUI screens and components. As such, all the screen controllers depend on the App class to function–shown by the depends-on relationship arrow.

STATE DESIGN PATTERN

Using the level of the Customer class, the premise of the State design pattern is formed. This was implemented due to the varying fees each level adds to online purchases. From the UML class diagram of **Figure 2**, it can be seen that the concrete classes SilverCustomer, GoldCustomer, and PlatinumCustomer inherit the attributes of the abstract class CustomerLevelState. Each concrete class represents a state of the customer and has a specific fee and level. The customer’s state will change in accordance with the balance, leading to a change in the fee and level.

JAVADOC COMMENTS AND PROCEDURAL ABSTRACTION

The class that contains javadoc comments and procedural abstraction is the User class. The User class encapsulates the requirements as stated in point two of the project manual. It contains the overview clause, abstraction function and rep invariant. Each method contains the necessary effects, modifies, and requires clauses as javadoc comments. Additionally, the abstraction function and rep invariant are implemented in the toString() and repOK() methods respectively.