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)	2023/2024 Semester 2

Instructor:	Boujemaa Guermazi

Assignment/Lab Number:	Project
Assignment/Lab Title:	Bank Account Application

Submission Date:	03/25/2024
Due Date:	03/25/2024

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Cheng	Vincent	501173673	07	[illegible]

*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/content/dam/senate/policies/pol60.pdf

The "Authentication" use case involves the interaction between the Customer and Manager actors and the system. It extends the login functionality for both the Customer and Manager. The system initiates this use case when either actor logs in by providing their username and password. Upon entry, the system verifies the login credentials to ensure they meet specified parameters. The Manager actor, upon successful authentication, gains the authority to add or delete a customer. When adding a customer, the Manager is prompted to create an account with an initial balance of at least $100. After authentication, the Customer actor gains access to the Transactions page, where options such as Deposit, Withdrawal, Online Purchase, and Logout are available. If the Customer proceeds with an Online Purchase, they are presented with options to buy a car, bike, or hat. Before checkout, the system checks the account balance to ensure it covers the purchase cost.

Name of Use Case	Authenticate
Actors	**Customer**: The individual who owns the bank account and interacts with the system to perform banking operations. **Manager**: Responsible for managing customer accounts, including adding or deleting customers from the system.
Entry condition	This use case starts when the Customer or Manager attempts to log in by providing their username and password.
Flow of Events	1. Customer or Manager enters their username and password. 2. System verifies the login credentials to ensure they meet specified parameters. 3. If credentials are valid: a. For Customer: Customer gains access to account-related functionalities b. For Manager: Manager gains authority to manage customer accounts.
Exit condition	This use case terminates when the Customer or Manager successfully logs in and completes their desired actions, or chooses to log out.
Exceptions	If the login credentials are invalid: System prompts the user to re-enter their

	credentials.
Special Requirements	Constraints: Each username must be unique

The Class Diagram illustrates the structure and relationships between the various classes in the bank account application. The main classes involved are Customer and Manager, representing the two types of profiles in the system. Each Customer object is associated with an Account, which encapsulates functionalities such as login, deposit, withdrawal, online purchases and log out. The Manager class has access to actions like login, logout, adding customers, and deleting customers. The Account class includes methods to determine the status of the account, which can be Silver, Gold, or Platinum, based on the balance. The relationships between classes depict how each component interacts within the system to facilitate banking operations and management functionalities.

The selected class to fulfill the outlined requirements is the Account class. Within the JavaDoc comments of this class, an Overview clause is provided, delineating the class's responsibilities and specifying its mutability status. Additionally, the abstraction function and the rep invariant are documented to ensure clarity regarding the class's behavior and constraints. Each method within the Account class is annotated with the necessary clauses such as effects, modifies, and requires, clarifying their functionality and preconditions. Also, the abstraction function is implemented within the toString() method and the rep invariant is enforced within the repOK() method.

The State design pattern is applied within the Level hierarchy and its subclasses Silver, Gold, and Platinum. Each subclass represents a different state of the account, and they encapsulate the behavior associated with transitioning between these states. Specifically, the changeLevel method in each subclass determines the logic for transitioning the account to a different level based on its current balance. This approach allows the Account class to alter its behavior dynamically at runtime, depending on its internal state, which aligns with the intent of the State design pattern. Additionally, the onlinePurchase method in each subclass adjusts the behavior of online purchases based on the account's current level, further demonstrating the flexibility and adaptability of the State pattern in managing the account's behavior transitions.