Toronto Metropolitan University

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

Course Title:	COE528
Course Number:	COE528
Semester/Year (e.g.F2016)	W2024

Instructor:	Boujemaa Guermazi

Assignment/Lab Number:	Project
Assignment/Lab Title:	Project

Submission Date:	March 23, 2024
Due Date:	March 23, 2024

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Patel	Owais	501181250	8	Owais

*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

Use Case Diagram

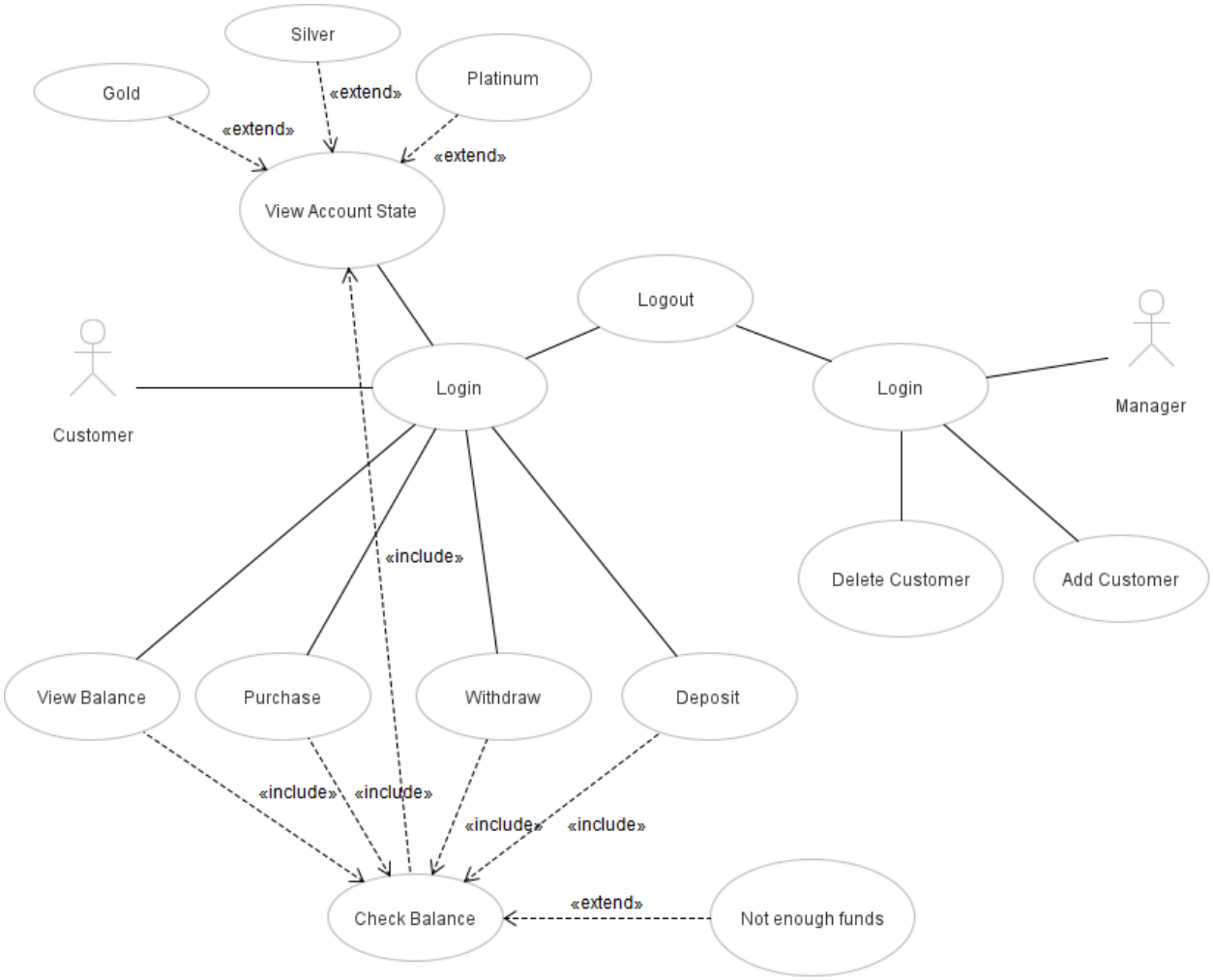

Figure 1: UseCaseDiagram of the Bank Application

Description:

The use case diagram shows the interactions in the bank application, with two main actors: Customer and Manager. Customers can log in to view their balance and make transactions, such as deposits, withdrawals, and purchases, with the "Check Balance" use case included to check if enough funds are available in the account. Customers can also log out. Managers can log in and perform administrative actions like adding or deleting customers, and they can also log out. The diagram shows "extends" relationships for different account states (Silver, Gold, Platinum) extending from the "View Account State" use case, as there are different fees for the different account states when making a purchase. The account states are checked each time a transaction is made to verify which state it is in.

The "Purchase" use case specifically allows a Customer to make a purchase, deducting the corresponding amount from their bank account. This use case includes the "Check Balance" use case, ensuring the customer has enough funds before processing the transaction. It also extends

to a "Not enough funds" use case, which might handle scenarios where the transaction cannot proceed due to insufficient account balance, providing the customer with an alert. The “Check Balance” case also includes the “View Account State”, as different purchase fees apply depending on the state(Silver, Gold, Platinum) they are in, which is verified when the transaction is made.

Class Diagram

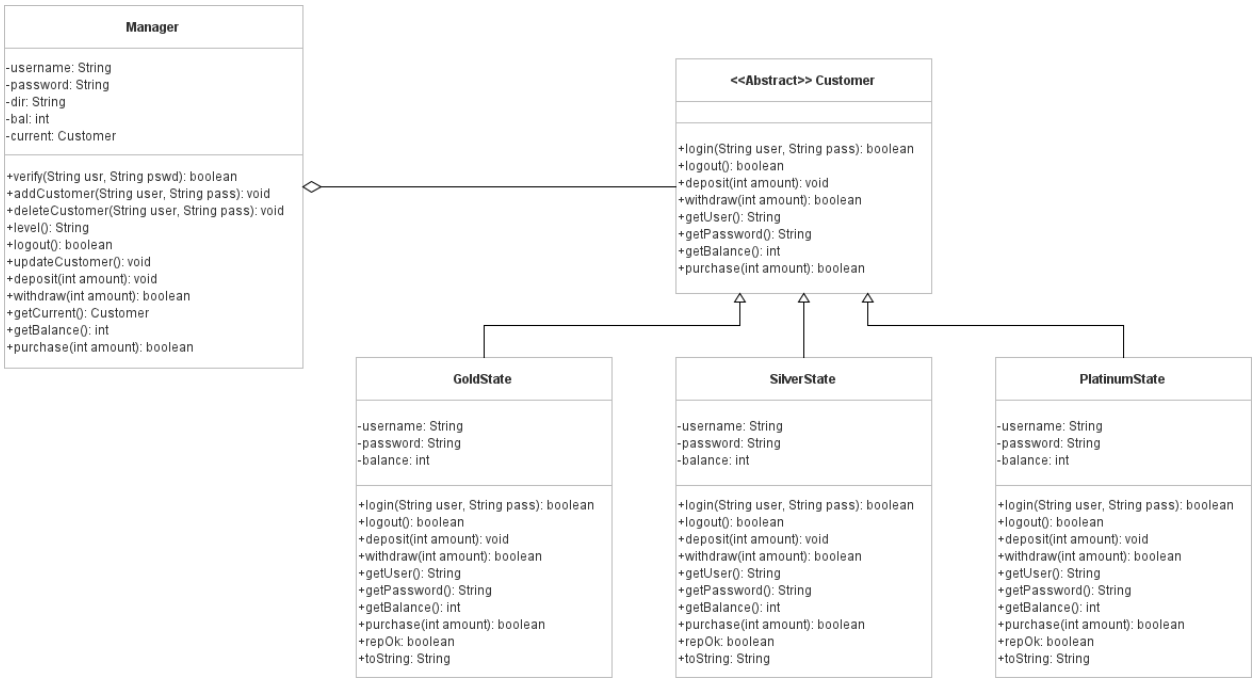

Figure 2: ClassDiagram of the Bank Application

Description:

The class diagram displays the structure of the bank application with four primary classes: “Manager”, an abstract “Customer” class, and “GoldState”, “SilverState”, and “PlatinumState” classes that inherit from “Customer”. The “Manager” class has attributes to store a username, password, a directory for customer files, a balance, and a reference to the current customer. Its methods include operations like verifying login credentials, adding and deleting customers, retrieving and updating customer levels, and also contains methods for conducting transactions (deposit, withdraw, purchase) but does not actually conduct these operations. The reason for these methods in the Manager class is that the customer state is managed within the Manager class by the updateCustomer method after a deposit. For example, a deposit might change a customer's state from SilverState to GoldState, and this transition is managed within the Manager class by the updateCustomer method after a deposit. It is also for file handling, abstraction, and encapsulation. The `Customer` class is an abstract class which defines the blueprint for customer operations such as login, logout, deposit, withdraw, and purchase, along with getters for username, password, and balance. The “GoldState”, “SilverState”, and “PlatinumState” classes implement these methods and have attributes for username, password, and balance, which are

common across all three. They also have “repOk” methods to ensure the object state is consistent, and “toString” methods for text representation. The “Manager” class has an aggregation relationship with “Customer”, implying that a manager manages customers but does not have exclusive ownership over them. The classes representing customer states (“GoldState”, “SilverState”, “PlatinumState”) indicate different levels of customers in the banking system with specific purchase fees for each account state.

The classes that I selected to address point number 2 mentioned in the lab manual is the Silver, Gold, and Platinum classes. Here, I included the “EFFECTS, MODIFIES, and REQUIRES” for each method. The rep invariant and abstraction function were also implemented in the repOk() and toString() methods respectively. All of these comments were made as javadoc comments. The overview clause was also written at the beginning of the class.