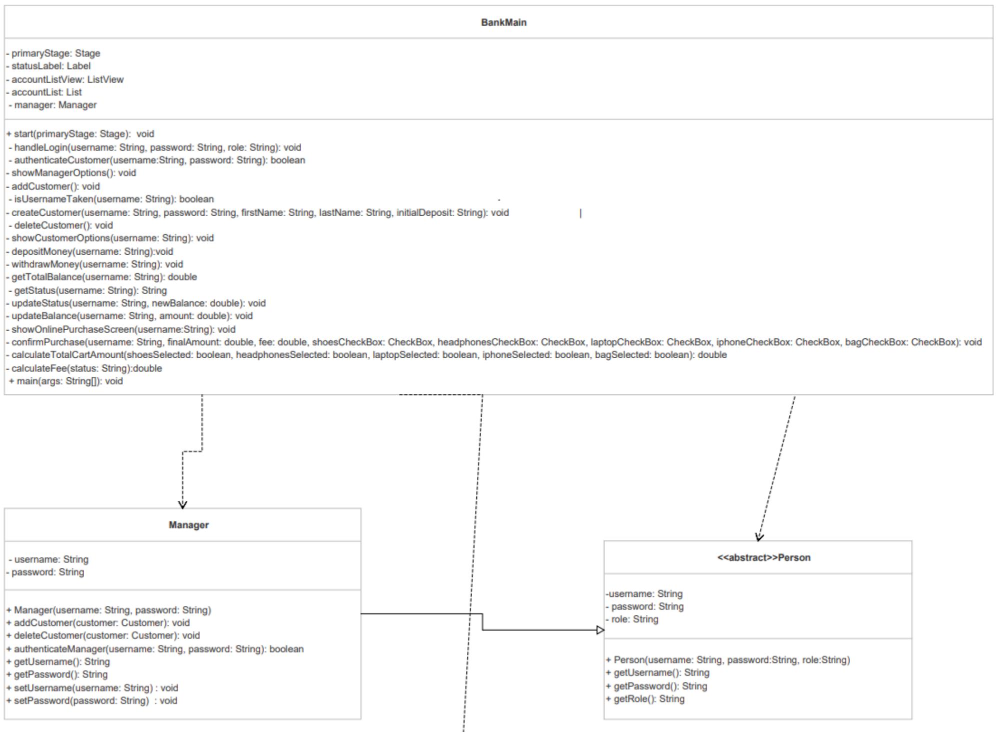
Alishba Adil
501160816
Section 12

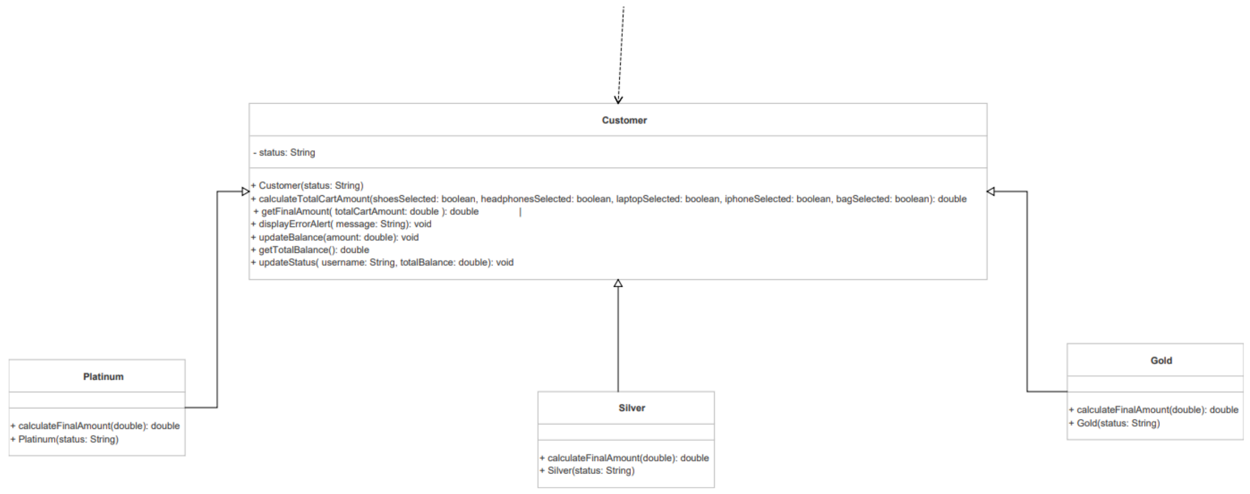
COE 528 Project Report

Class Diagram:

The class diagram below shows the overall relationship between the 7 different classes in my bank application project which are Manager, Person, Customer, BankMain, Platinum, Gold and Silver. At the center is the Person class, serving as a template defining user entities, encompassing both customers and managers such as username, login and role. The Customer and Manager classes are tailored to fulfill distinct roles within the system. The manager and customer class extend the person class implementing person methods and adding new ones. Manager adds additional methods to add and delete customers as well as define the username and password for Manager login. The Customer class, manifesting in subclasses such as Silver, Gold, and Platinum, designed to handle nuanced account status-specific functionalities, which are the computation of final purchase amounts. The BankMain class implements the graphical user interface and backend operations, ensuring seamless user experiences. It manages all the user interface and the interactions including the logging in and out, the money transactions and all the customer management.Through working together effectively, these classes make the banking application run smoothly and allow users to interact with it easily.

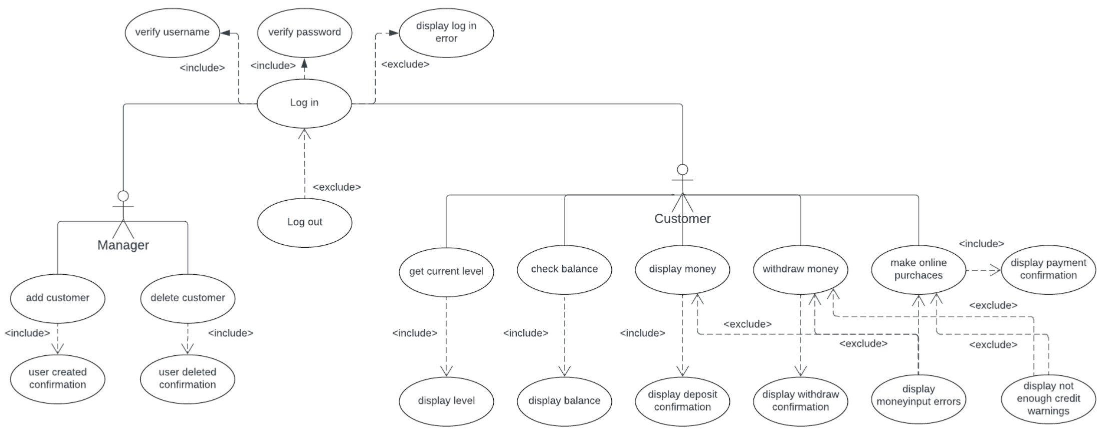

BankMain
- primaryStage: Stage
- statusLabel: Label
- accountListView: ListView
- accountList: List
- manager: Manager
+ start(primaryStage: Stage): void
- handleLogin(username: String, password: String, role: String): void
- authenticateCustomer(username:String, password: String): boolean
- showManagerOptions(): void
- addCustomer(): void
- isUsernameTaken(username: String): boolean
- createCustomer(username: String, password: String, firstName: String, lastName: String, initialDeposit: String): void
- deleteCustomer(): void
- showCustomerOptions(username: String): void
- depositMoney(username: String):void
- withdrawMoney(username: String): void
- getTotalBalance(username: String): double
- getStatus(username: String): String
- updateStatus(username: String, newBalance: double): void
- updateBalance(username: String, amount: double): void
- showOnlinePurchaseScreen(username:String): void
- confirmPurchase(username: String, finalAmount: double, fee: double, shoesCheckBox: CheckBox, headphonesCheckBox: CheckBox, laptopCheckBox: CheckBox, iphoneCheckBox: CheckBox, bagCheckBox: CheckBox): void
- calculateTotalCartAmount(shoesSelected: boolean, headphonesSelected: boolean, laptopSelected: boolean, iphoneSelected: boolean, bagSelected: boolean): double
- calculateFee(status: String):double
+ main(args: String[]): void
Manager
- username: String
- password: String
+ Manager(username: String, password: String)
+ addCustomer(customer: Customer): void
+ deleteCustomer(customer: Customer): void
+ authenticateManager(username: String, password: String): boolean
+ getUsername(): String
+ getPassword(): String
+ setUsername(username: String) : void
+ setPassword(password: String) : void
<<abstract>>Person
-username: String
- password: String
- role: String
+ Person(username: String, password:String, role:String)
+ getUsername(): String
+ getPassword(): String
+ getRole(): String

Alishba Adil
501160816
Section 12



Customer
- status: String
+ Customer(status: String)
+ calculateTotalCartAmount(shoesSelected: boolean, headphonesSelected: boolean, laptopSelected: boolean, iphoneSelected: boolean, bagSelected: boolean): double
+ getFinalAmount(totalCartAmount: double): double
+ displayErrorAlert(message: String): void
+ updateBalance(amount: double): void
+ getTotalBalance(): double
+ updateStatus(username: String, totalBalance: double): void
Platinum
+ calculateFinalAmount(double): double
+ Platinum(status: String)
Silver
+ calculateFinalAmount(double): double
+ Silver(status: String)
Gold
+ calculateFinalAmount(double): double
+ Gold(status: String)

Case Diagram:



Name of Use Case: Online Purchase
Actors: Customer, Manager
Entry condition: This use case starts when a customer decides to make an online purchase through the banking system.
Flow of Events:
The customer logs into the system using their username and password.
The customer selects items they want to purchase by checking corresponding checkboxes (e.g., shoes, headphones, laptop, iPhone, bag).
The customer clicks on the "Purchase" button.
The system calculates the total amount of the selected items.
The system determines the final amount to be charged, considering the customer's status (Gold, Silver, or Platinum).
If the total cart amount is less than $50, the system displays an error alert.
If the customer's balance is sufficient, the system proceeds with the purchase.
The system deducts the final amount from the customer's account balance.

If necessary, the system updates the customer's status based on the new balance.
The system confirms the successful completion of the transaction.
Exit condition:
This use case terminates when the online purchase transaction is completed successfully.
Exceptions:
If the total cart amount is less than $50, the system displays an error alert indicating that the total should be over $50.
If the customer's balance is insufficient for the transaction, the system displays an error alert indicating the insufficient balance.
Special Requirements:
Nonfunctional Requirements: The system should provide a user-friendly interface for customers to make purchases efficiently.
Constraints: The system should ensure secure transactions and accurate balance updates.

Class Overview

Class Selected **Manager:**
The Manager class represents a manager in the banking system, who manages customer-related tasks. It holds the manager's username and password securely and cannot be changed once set. The abstraction function helps understand how the class works by mapping its internal state to its purpose as a manager. The repOk method checks if the class is properly set up, ensuring both username and password are not empty.
Overview is as follows:
Manager class represents a manager in the system. Managers have the responsibility
of managing customers and their accounts. This class is mutable as it provides methods
to add and delete customers from the system.
Abstraction Function:
- Represents a manager with a username and a password.
Rep Invariant:
- The username and password must not be null.
Effects Clause:
- addCustomer(Customer customer): Adds the specified customer to the system.
- deleteCustomer(Customer customer): Deletes the specified customer from the system.
- authenticateManager(String username, String password): Checks if the provided username and password match the manager's credentials.
Modifies Clause:
- addCustomer(Customer customer): Modifies the system by adding a new customer.
- deleteCustomer(Customer customer): Modifies the system by removing a customer.
Requires Clause:
- authenticateManager(String username, String password): Requires valid username
and password parameters to authenticate the manager.

State Design Pattern

Alishba Adil
501160816
Section 12

State design pattern is applied through the use of concrete state classes (Silver, Gold, Platinum) to encapsulate the behavior and state transitions associated with different customer levels. Each state class extends a class, defining methods for calculating final amounts, updating balances, and handling specific behaviors unique to the customer level. The Customer class serves as the context class, maintaining the current state of the customer and delegating behavior to the current state object. Methods in the Customer class call corresponding methods in the current state object, allowing for dynamic behavior based on the customer's level. The online purchase amount is calculated in customer class depending on what class the user is currently in. For instance, if the user is in gold, it calls the gold class where the total amount is calculated based with a fee of $10. State transitions, such as upgrading or downgrading a customer's level, are handled within the Customer class through methods from Bank Main Class.