DOKUZ EYLÜL UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CME 2210
Object Oriented Analysis and Design

BANK MANAGEMENT SYSTEM

by
2020510055 Mahmut Yiğit Kılınççıoğlu
2020510059 Eren Kötüğ

CHAPTER ONE

INTRODUCTION

SCOPE OF PROJECT

This is a bank application for Windows PC users. Main goal of the project is develop a bank management software to perform various simple banking tasks. Team consists of 2 members(Eren Kötüğ, Mahmut Yiğit Kılınççıoğlu). Estimated time required to complete this project is two months. By April 2024, project is expected to be completed.

PROJECT DESCRIPTION

The Bank Management System is a software application designed to facilitate the management of banking operations. It provides functionalities to manage customer accounts, transactions, loans, and other banking-related activities efficiently. The primary goal of this project is to create a robust and user-friendly system that automates various banking tasks, thereby enhancing the overall efficiency and accuracy of banking operations.

TASKS TO COMPLETE

- Create a new customer account
- Update customer information such as name, address, or contact number
- View details of customer accounts
- Perform transactions such as deposits and withdrawals for customer accounts
- Exchange with the current exchange rates
- Money transfer between customers

TECHNICAL REQUIREMENTS

- JAVA
- JAVA SWING

CHAPTER TWO

REQUIREMENTS

CLASSES

CustomerInterface

Functions in this class:

- String getCustomerID()
- void setCustomerID(String customerID)
- String getPassword()
- void setPassword(String password)
- String getName()
- void setName(String name)
- String getSurName()
- void setSurName(String surName)
- String getAddress()
- void setAddress(String address)
- String getContactNumber()
- void setContactNumber(String contactNumber)

- void createNewAccount(Account account, Integer accountType)
- Map<Integer, Account> getAccounts()
- void transferMoney(Customer receiver, double amount)

Customer

The customer will able to exchange and transfer money between another customer. He will be able to buy currency.

Attributes in this class:

- private String customerID
- private String password
- private String name
- private String surName
- private String address
- private String contactNumber
- private Map<Integer,Account> accounts

Functions in this class:

- public GetSetMethods()
- public Customer(String customerID, String password, String name, String surName, String address, String contactNumber)
- public createNewAccount(Account account ,Integer accountType)
- public void transferMoney(Customer reciever ,double amount)

Bank

Attributes in this class:

- private HashMap<String, Customer> bank

Functions in this class:

- public HashMap<String, Customer> getBank()

AccountInterface

Functions in this class:

- String getAccountID()
- void setAccountID(String accountID)
- double getBalance()
- void setBalance(double balance)
- String getCurrencyUnit()
- void setCurrencyUnit(String currencyUnit)
- HashMap<String, Double> getExchangeRates()
- void deposit(double amount)
- void withdraw(double amount)

Account

Attributes in this class:

- protected String accountID
- protected double balance
- protected String currencyUnit
- protected HashMap<String, double> exchangeRates

Functions in this class:

- public Account(String accountID)
- public GetSet Methods()
- public void deposit(double amount)
- public void withdraw(double amount)

CurrencyAccount

This class will extend Account Class

Attributes in this class:

- private int accountType

Functions in this class:

- public CurrencyAccount(String accountID)

  This function calls its superclass which is Account class.

- public GetSetMethods()

- public double exchangeCurrency(String fromCurrency, String toCurrency, double amount)

DepositAccount

This class will extend Account Class

Attributes in this class:

- private int accountType

Functions in this class:

- public DepositAccount(String accountID)

  This function calls its superclass which is Account class.

- public GetSetMethods()

SavingAccount

This class will extend Account Class

Attributes in this class:

- private int accountType

Functions in this class:

- public SavingAccount(String accountID)

  This function calls its superclass which is Account class.

- public GetSetMethods()

SignUpPage

Attributes in this class:

- private JFrame frame
- private JButton signUpButton
- private JTextField userIDField
- private JPasswordField userPasswordField
- private JTextField nameField
- private JTextField surnameField
- private JTextField addressField
- private JTextField contactNumberField
- private JLabel userIDLabel
- private JLabel userPasswordLabel
- private JLabel nameLabel
- private JLabel surnameLabel
- private JLabel addressLabel
- private JLabel contactNumberLabel
- private JLabel messageLabel

Functions in this class:

- SignUpPage(HashMap<String, Customer> bank)
- public HashMap<String, Customer> getBank()

LoginPage

Attributes in this class:

- private JFrame frame
- private JButton loginButton
- private JButton resetButton
- private JButton signupButton
- private JTextField userIDField
- private JPasswordField userPasswordField
- private JLabel userIDLabel
- private JLabel userPasswordLabel
- private JLabel messageLabel
- private HashMap<String, Customer> bank

Functions in this class:

- public LoginPage(HashMap<String, Customer> bank)

TransferMoneyPage

Attributes in this class:

- private JFrame frame
- private JTextField accountNumberField
- private JTextField amountField

Functions in this class:

- public TransferMoneyPage(HashMap<String, Customer> bank, String customerID)
- private void transferMoney(String recieverID, double amount)

DepositWithdrawPage

Attributes in this class:

- private JFrame frame
- private JRadioButton depositRadioButton
- private JRadioButton savingRadioButton
- private JRadioButton currencyRadioButton
- private JTextField amountField
- private final String TITLE = "DEUBank"

Functions in this class:

- public DepositWithdrawPage(Customer currentCustomer)

CreateAccountPage

Attributes in this class:

- private JFrame frame
- private JRadioButton depositRadioButton
- private JRadioButton savingRadioButton
- private JRadioButton currencyRadioButton
- Customer currentCustomer
- private final String TITLE = "DEUBank"

Functions in this class:

- public CreateAccountPage(Customer currentCustomer)
- private void createAccount(Customer currentCustomer)

UpdateInfoPage

Attributes in this class:

- private JFrame frame
- private JTextField nameField
- private JTextField surnameField
- private JTextField addressField
- private JTextField contactNumberField
- private JButton updateButton
- private JPanel rightPanel
- private Customer currentCustomer
- private HashMap<String, Customer> bank

Functions in this class:

- public UpdateInfoPage(Customer currentCustomer, HashMap<String, Customer> bank, JPanel rightPanel)
- private void updateRightPanel()

ShowBalancePage

Attributes in this class:

- private JFrame frame
- private Customer currentCustomer

Functions in this class:

- public ShowBalancePage(Customer currentCustomer)
- private void showBalance(JTextArea accountListArea)

ExchangeCurrencyPage

Attributes in this class:

- private JFrame frame
- private JLabel balanceLabel
- private JComboBox<String> toCurrencyBox
- private JButton exchangeButton
- private final String TITLE = "DEUBank"
- private String currencyUnit
- private HashMap<String, Double> exchangeRates
- private Customer currentCustomer
- private double currentBalance

Functions in this class:

- public ExchangeCurrencyPage(Customer currentCustomer)
- private void exchangeCurrency()

MainPage

Attributes in this class:

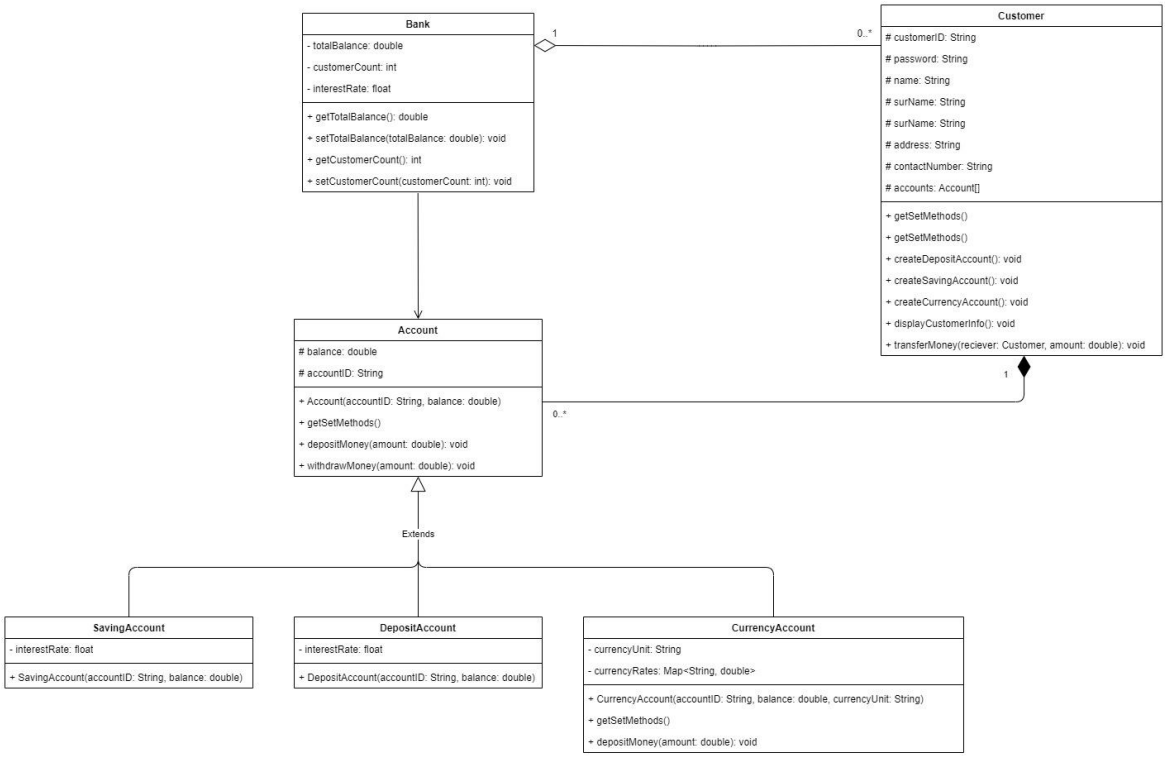
- private JFrame frame
- private Customer currentCustomer
- private final String TITLE = “DEUBank”

Functions in this class:

- public MainPage(HashMap<String, Customer> bank, String CustomerID)

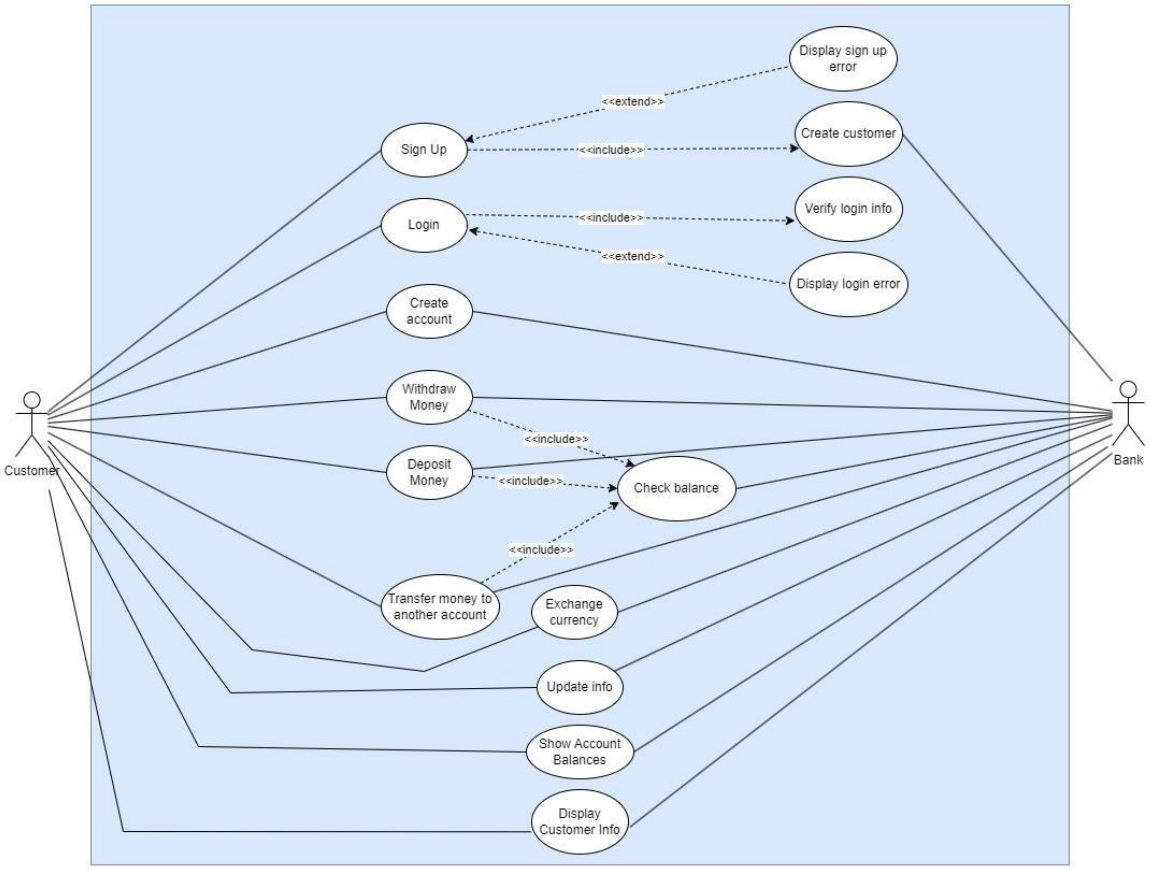
CHAPTER THREE

CLASS DIAGRAM



In the class diagram, the central Bank class is fundamental, acting as the core entity. Customers are linked to the bank, indicating their dependency; without the bank, customers cannot exist within the system. Each customer has the capability to hold multiple accounts; for instance, a customer may possess both a deposit account and an exchange account simultaneously. However, every account is uniquely associated with only one customer. Furthermore, deposit, saving and currency accounts are subclasses of the Account class.

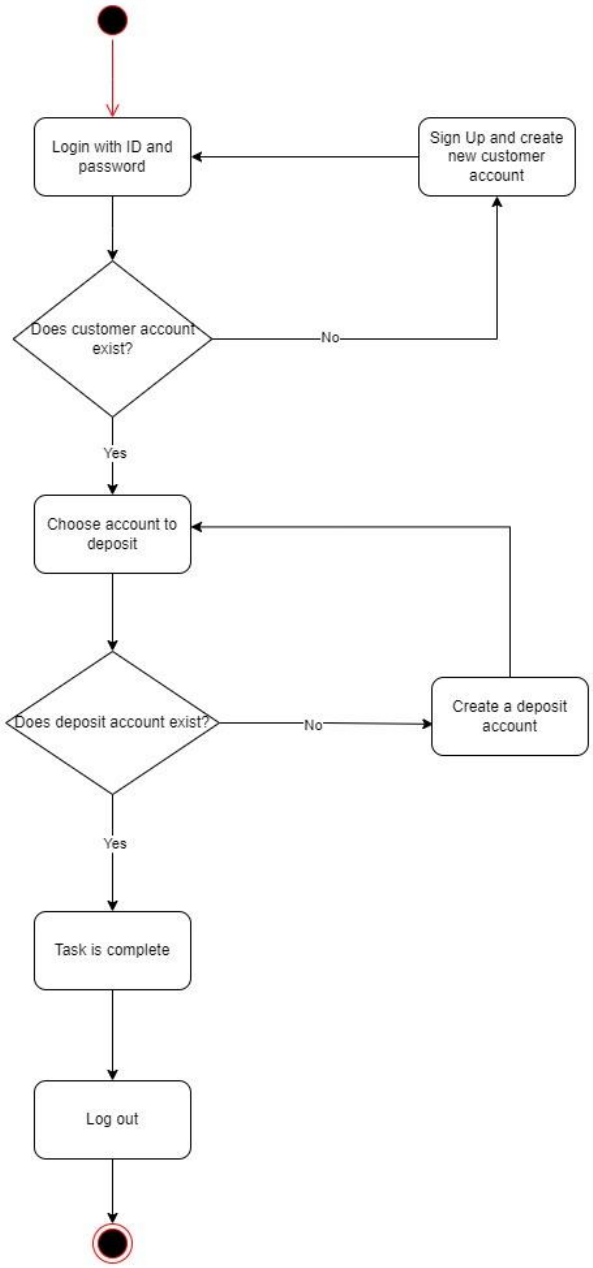
USE-CASE DIAGRAM



Our bank application's use case diagram outlines the following functionalities:

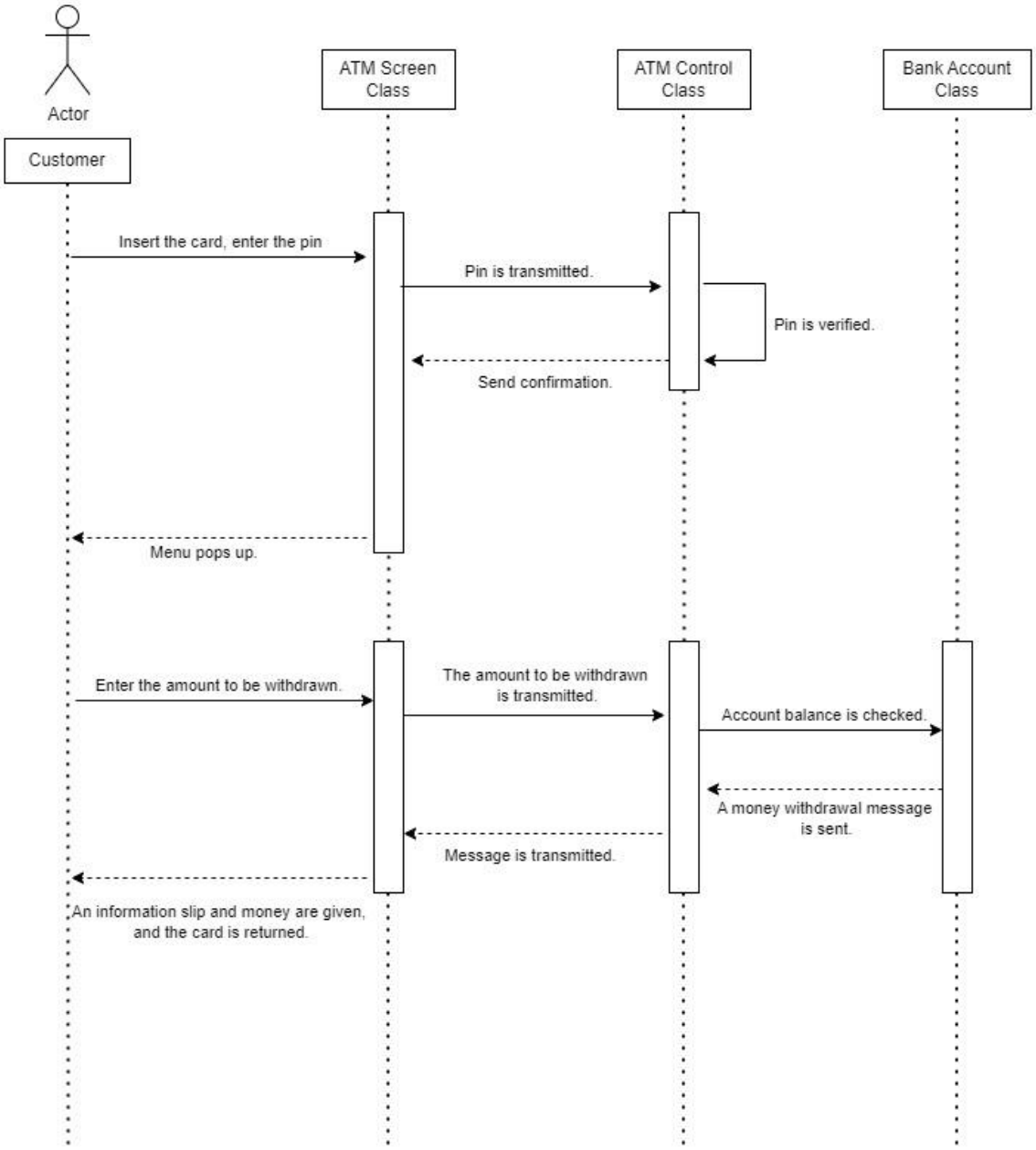
- Customers can deposit and withdraw money from their deposit, saving and current accounts, necessitating the availability of such accounts for deposit and withdrawals.
- To login, customer has to sign up and create a new customer account.
- Customers can create accounts such as deposit, saving and currency account.
- Logged-in customers can access their account information.
- To engage in currency exchange, customers must log in and create a currency account.
- To execute operations deposit-withdraw and transfer, customer has to have enough balance in their accounts.
-

ACTIVITY DIAGRAM



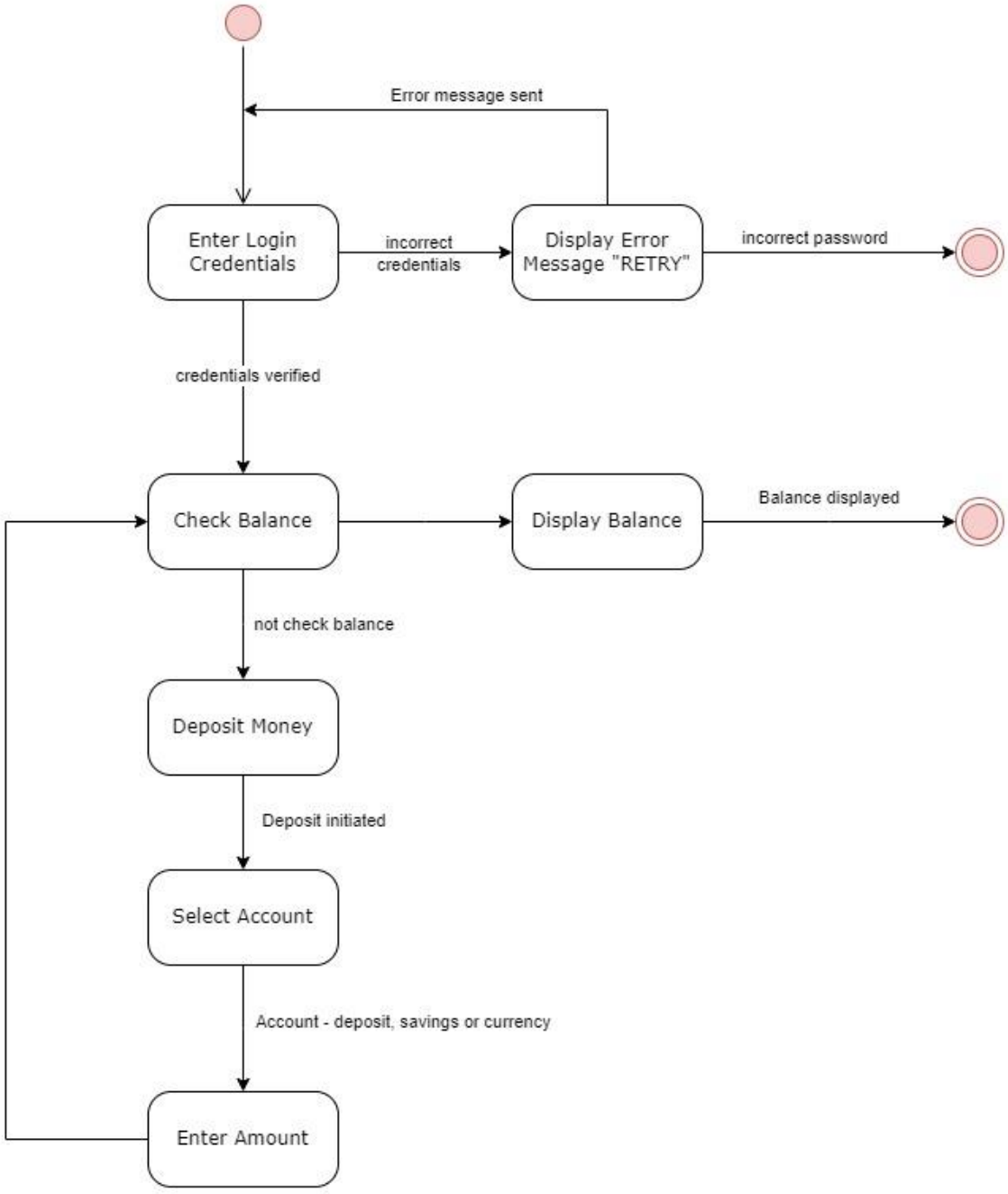
This is the activity diagram of creating a deposit account statement. User logins to the system with necessary controls and creates a deposit account successfully.

SEQUENCE DIAGRAM



This is the sequence diagram of the withdraw operation. Customer logins the system with correct PIN and gets verified, the menu pops up. Customer chooses the operation to be executed, and the sequence is completed.

STATE DIAGRAM



This is the state diagram of checking balance and depositing money to the selected account. Customer's login info is checked, if it's wrong repeatedly the state reaches dead state. If login is successful and the customer only wants to check balance, it again reaches dead state. If customer wants to deposit money, first he needs to check balance and then he can continue with selecting the account and then he completes with checking balance of deposited account, then the state is over.

CHAPTER FOUR

IMPLEMENTATION

Customer.java

```
public class Customer implements CustomerInterface {
    private String customerID;
    private String password;
    private String name;
    private String surName;
    private String address;
    private String contactNumber;
    private Map<Integer, Account> accounts;
     public Customer(String customerID, String password, String name, String surName,
String address,
            String contactNumber) {
        this.customerID = customerID;
        this.password = password;
        this.name = name;
        this.surName = surName;
        this.address = address;
        this.contactNumber = contactNumber;
        this.accounts = new HashMap<>();
    }
    public String getCustomerID()
    public void setCustomerID(String customerID)
    public String getPassword()
    public void setPassword(String password)
    public String getName()
    public void setName(String name)
```

```
    public String getSurName()

    public void setSurName(String surName)

    public String getAddress()

    public void setAddress(String address)

    public String getContactNumber()

    public void setContactNumber(String contactNumber)

    public void createNewAccount(Account account, Integer accountType) {
       accounts.put(accountType, account);
    }

    public Map<Integer, Account> getAccounts()

    public void transferMoney(Customer reciever, double amount) {
       // this if-else block controls if both sender and reciever has deposit accounts
       // and proceeds to transfer operation
       if (!reciever.getAccounts().containsKey(1) || !this.accounts.containsKey(1)) {

       } else if (this.accounts.get(1).getBalance() >= amount) {
          double senderBalance = this.accounts.get(1).getBalance();
          this.accounts.get(1).setBalance(senderBalance - amount);
          double recieverBalance = reciever.accounts.get(1).getBalance();
          reciever.accounts.get(1).setBalance(recieverBalance + amount);

       }
    }
}
```

The Customer class represents individual customers of the bank. Each customer has unique attributes such as ***customerID, password, name, surname, address***, and ***contactNumber***. Additionally, a Customer object maintains a collection of accounts associated with that customer through a ***Map<Integer, Account>***. This allows for easy access to all accounts owned by a particular customer. **Map** is initiated by constructor.

transferMoney(Customer receiver, double amount): This method enables a customer to transfer a specified amount of money to another customer. It utilizes the receiver parameter to identify the recipient of the funds and the amount parameter to specify the sum being transferred. The implementation ensures that the transfer is only possible if the sender has sufficient funds in their account. ***This method is used inside the TransferMoneyPage.java.***

Bank.java

```
public class Bank {
    private HashMap<String, Customer> bank;
    public Bank() {
        bank = new HashMap<String, Customer>();
    }
    public HashMap<String, Customer> getBank() {
        return bank;
    }
}
```

The Bank class serves as a container for managing customer data. It employs a ***HashMap<String, Customer>*** to store customers, with each entry keyed by the customer's unique ***customerID***. This data structure facilitates efficient retrieval and manipulation of customer information.

AccountInterface.java

```
public interface AccountInterface {
    String getAccountID();
    void setAccountID(String accountID);
    double getBalance();
    void setBalance(double balance);
    String getCurrencyUnit();
    void setCurrencyUnit(String currencyUnit);
    HashMap<String, Double> getExchangeRates();
```

```
    void deposit(double amount);
    void withdraw(double amount);
}
```

This is the interface of **abstract class Account.**

Account.java

```
public class Account implements AccountInterface {
    protected String accountID;
    protected double balance;
    protected String currencyUnit;
    protected HashMap<String, Double> exchangeRates;

    public Account(String accountID) {
        this.accountID = accountID;
        this.balance = 0;
        this.currencyUnit = "TL";
    }

    public HashMap<String, Double> getExchangeRates() {
        return exchangeRates;
    }

    public String getCurrencyUnit() {
        return currencyUnit;
    }

    public void setCurrencyUnit(String currencyUnit) {
        this.currencyUnit = currencyUnit;
    }
```

```
    public String getAccountID() {
        return accountID;
    }

    public void setAccountID(String accountID) {
        this.accountID = accountID;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            this.balance += amount;
        } else {
            System.out.println("Deposit amount must be positive.");
        }
    }

    public void withdraw(double amount) {
        if (this.balance >= amount)
            this.balance -= amount;
        else
            System.out.println("There is not enough money in your account.");
    }
}
```

The Account class is an abstract superclass that defines common methods and attributes shared by **DepositAccount, CurrencyAccount, SavingAccount** in the banking system. It encapsulates essential functionalities such as ***deposit*** and ***withdraw***, allowing for the addition and removal of funds from an account. Additionally, it includes methods for accessing and modifying attributes such as **balance** and **currencyUnit**. Accounts' default **currencyUnit** is "TL", but because of the functionality of currency exchange, CurrencyAccount's unit might change according to ***exchangeRates***.

deposit(double amount): Adds the specified amount of money to the account's balance.
withdraw(double amount): Subtracts the specified amount of money from the account's balance. ***These methods are called inside DepositWithdrawPage.java.***

DepositAccount.java

```
public class DepositAccount extends Account {
   private int accountType;

   public DepositAccount(String accountID) {
      super(accountID);
      this.accountType = 1;
   }
   public int getAccountType() {
      return accountType;
   }
}
```

The DepositAccount class is a subclass of **Account** class. Constructor calls it's superclass constructor and accountType is added as 1. ***Account type*** is an attribute used to determine the type of accounts.

SavingAccount.java

```
public class SavingAccount extends Account {
```

```
    private int accountType;
    public SavingAccount(String accountID) {
        super(accountID);
        this.accountType = 2;
    }
    public int getAccountType() {
        return accountType;
    }
}
```

The SavingAccount class is a subclass of **Account** class. Constructor calls it's superclass constructor and accountType is added as 2. ***Account type*** is an attribute used to determine the type of accounts.

CurrencyAccount.java

```
public class CurrencyAccount extends Account {
    private int accountType;
    public CurrencyAccount(String accountID) {
        super(accountID);
        this.accountType = 3;
        this.exchangeRates = new HashMap<>();
        exchangeRates.put("TL", 1.0); // Base currency
        exchangeRates.put("USD", 0.031);
        exchangeRates.put("EUR", 0.028);
        exchangeRates.put("GBP", 0.024);
        exchangeRates.put("JPY", 4.83);
    }
    public int getAccountType() {
        return accountType;
    }
    public String getCurrencyUnit() {
        return currencyUnit;
```

```
    }
    public void setCurrencyUnit(String currencyUnit) {
        this.currencyUnit = currencyUnit;
    }

     public double exchangeCurrency(String fromCurrency, String toCurrency, double amount) {
                                    if (!exchangeRates.containsKey(fromCurrency) || !exchangeRates.containsKey(toCurrency)) {
            throw new IllegalArgumentException("Invalid currency");
        }
        double fromRate = exchangeRates.get(fromCurrency);
        double toRate = exchangeRates.get(toCurrency);

        // Calculate the exchanged amount
        return (amount / fromRate) * toRate;
    }
}
```

The CurrencyAccount class is a subclass of **Account** class. Constructor calls it's superclass constructor and ***accountType*** is added as 3. ***HashMap<String, Double> exchangeRates*** is initiated. ***Account type*** is an attribute used to determine the type of accounts.

exchangeCurrency(String fromCurrency, String toCurrency, double amount): This method is used for exchanging between currencies. It takes the current currency as ***fromCurrency*** and takes input as ***toCurrency***, and accounts balance as ***amount***. ***The method is called inside ExchangeCurrencyPage.java*** and the exchange operation is executed.

Test.java

The initiation class of the application.

These are the main operational classes for necessary banking functions such as tranfering Money, exchanging currency, creating accounts, deposit-withdraw operations. The classes used for GUI and login-signup operations are below.

The libraries and elements used for GUI are:

```
import javax.swing
import java.awt
import java.awt.event.ActionEvent
import java.awt.event.ActionListener
import java.awt.Color
import java.awt.Font
import javax.swing.JButton
import javax.swing.JFrame
import javax.swing.JLabel
import javax.swing.JOptionPane
import javax.swing.JPasswordField
import javax.swing.JTextField
```

SignUpPage.java

This class contains the necessary java swing elements. Its primary function is to take the required data as input when creating a user account and place it into the bank ***hashMap***. It also validates the inputs entered by the user while entering their information.

LoginPage.java

This class is the initial one that opens when the application starts and initializes the bank ***hashMap***. It contains a button that calls the ***SignUp*** constructor. Login process occurs here. During the login process, it checks whether the entered ***customerID*** and ***password*** exist in the bank. If they are not found, an error pop-up is displayed.

CreateAccountPage.java

The constructor of this class receives a ***Customer*** object from the **MainPage** ***class***. This component, opened by clicking a button on the **MainPage**, allows the user to create deposit, saving, and currency accounts. These operations are performed using swing elements such as radio buttons and buttons.

Methods:

private void CreateAccount(Customer currentCustomer)

DepositWithdrawPage.java

In this class, we perform deposit and withdraw operations using the ***Customer*** object provided in the constructor from **MainPage**. When we click the button on **MainPage**, a new window appears, prompting us to select an account and enter the amount. Then, by clicking the deposit or withdraw button, the transaction is executed. Deposit and withdraw methods are called from the **Account** class.

TransferMoneyPage.java

In the constructor of this class, a ***Customer*** object and the bank ***HashMap*** are provided from the **MainPage** class. This component, opened by clicking a button on the **MainPage**, allows the user to transfer money to another account. For this operation, the user needs to know the ID of the recipient account. This is done by calling the ***transferMoney*** method from the Customer class.

ExchangeCurrencyPage.java

This class called from **MainPage** to perform currency exchange operations. This page prompts the user to input the unit to which they want to convert their balance, and then calls a function to convert the account's balance to the specified unit, completing the operation.

Methods:

private void exchangeCurrency()

ShowBalancePage.java

We use the instance of this class to retrieve and display the user's account information to the user.

UpdateInfoPage.java

In this class, we provide input fields and buttons in a new window to allow the user to update their name, surname, address, and contact number information.

MainPage.java

In this class' constructor, the main page of bank application is created using swing elements like buttons, panels, frame etc. All of the operations are reached within this object. There are buttons for every component such as createAccount, DepositWithdraw, ExchangeCurrency, ShowBalance, UpdateInfo, TransferMoney. There is also a LogOut button which takes the user to the login page.

CHAPTER FIVE

CONCLUSION AND FUTURE WORKS

This project is a Java program representing the graphical user interface (GUI) of a banking management system. It provides functionalities such as creating new user, log in, create accounts, deposit and withdrawals, transferring money between accounts, updating user info using HashMap to store customer information. All tasks are completed.

To enchance the project's professionalism in the future, the following improvements can be made:

1.Database Integration: Storing customer and account data in a database would be a good improvement for this project.

2.Improved Error Handling: Effectively handle user errors and provide user-friendly messages for incorrect inputs.

3.Expanded Functionality: Adding more features to the project is going to lift this project to the next level.

4.Graphical Interface: Designing more user-friendly and appealing interface would be good improvement.