DOKUZ EYLÜL UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CME 2210
Object Oriented Analysis and Design

BANK APPLICATION

by
Ulaş Öncül
Hasan Mete Akdeniz
Batuhan Doğan

CHAPTER ONE

INTRODUCTION

SCOPE OF PROJECT

First off all, our poject is a bank application basicly. Our main goal is produce a minimal and effective bank application for Windows PC users. Three people (Ulaş Öncül, Hasan Mete Akdeniz and Batuhan Doğan) will work as a team to complete the project. When we complete our project completely, everyone using your bank application will be able to handle their basic work. We plan to finish our project in two months.

PROJECT DESCRIPTION

This project is mainly about a bank application that contains easy user interface. In first view of app there will be a menu that contains creating new account, login account, exchange rate etc. For inside of account section there will be a lot of options like deposit, withdrawals, transfering, exchange, investing money, loan calculater etc.

OBJECTIVES OF PROJECT

By April 2023, our team is going to develop a bank application, including an image templete. We will enhance the effortless sharing of information among the different departments by the end of the project’s first phase.

PROJECT SUCCESS CRITERIA

- Create a new customer account successfully
- Money transfer between users
- Calculating loan per users
- Customers' can view account information without error
- Exchange wıth the current exchange rate

TECHNIAL REQUIREMENTS

- ECLIPSE JAVA
- JAVA SWING

CHAPTER TWO

REQUIREMENTS

CLASSES

Customer

The customer will interact with its account. He will able to exchange and transfer money between another customers. He will able to buy currency.

Attributes in this class:

- Protected String id
- Protected String password
- Protected Account[]

Functions in this class:

- Public GetSetMethods()
- public int transferMoney(Customer reciever, double amount)
- public boolean sellCurrency(double amount)
- public void createSavingAccount()
- public void createAccount()

Account

Attributes in this class:

- Protected Double Balance
- protected String accountId
- protected ArrayList<String> loglar;

Functions in this class:

- public Account(String accountId, double balance
- public String getAccountId()
- Public Double GetBalance()
- public void depositMoney(double amount)
- public void withdraw(double amount)

SavingAccount

This class will extend Account class.

Attributes in this class:

- Protected constant Interest Rate

Functions in this class:

- public SavingAccount(String accountId, double balance)
- public void depositMoney(int amount)

- public void applyInterest()

Currency Account

This class will extends Account Class

Attributes in this class:

- private double EURO = 0.0467;
- private double DOLLAR = 0.050;
- private double STERLIN = 0.041
- private String currencyUnit

Functions in this class:

- public CurrencyAccount(String accountId, double balance, String currencyUnit)
- public void depositMoney(double amount)
- public String showBalance()

IdAndPasswords

Attributes in this class:

- private String[] ids private double DOLLAR = 0.050;
- private String[] passwords
- Customer [] a

- private HashMap<String, Customer> loginInfo = new HashMap<String, Customer>()

Functions in this class:

- public IdAndPasswords()
- public HashMap getLoginInfo()

WelcomePage

This class will extend JFrame

Attributes in this class:

- private JButton btnWithdraw
- private JButton btnApplyInterest
- private JButton btnBalance
- private JLabel lblAccountNumber
- private JLabel lblMessage;
- private JTextField txtAccountNumber

- private JTextField txtBalance
- private JRadioButton radWithdraw
- private JRadioButton radTransferMoney
- private ButtonGroup transGroup
- private String nameBank
- private JPanel inputPanel
- private HashMap<String, Customer> loginInfo = new HashMap<String,Customer>()

Functions in this class:

- public WelcomePage(HashMap<String, Customer> loginInfoOriginal,String idOfCustomer)public HashMap getLoginInfo()
- private void setupInputs()
- public void actionPerformed
- private void showInterest()
- private void deposit(double Amount)
- private void checkBalance()
- private boolean checkAccount(int accountType)
- private void createAccount()

SignUpPage

This class will implements from ActionListener

Attributes in this class:

- private JFrame frame
- private JButton resetButton
- private JButton signupButton private JLabel lblMessage;
- private JTextField userIDField private JTextField txtBalance
- private JPasswordField userPasswordField private JRadioButton radTransferMoney
- private JLabel userIDLabel private String nameBank

Functions in this class:

- public HashMap<String, Customer> getLoginInfo() private void setupInputs()
- public signUpPage(HashMap<String, Customer> loginInfoOriginal) private void showInterest()
- public void actionPerformed(ActionEvent e) private void checkBalance()

LogInPage

This class will implements from ActionListener

Attributes in this class:

- private JFrame frame

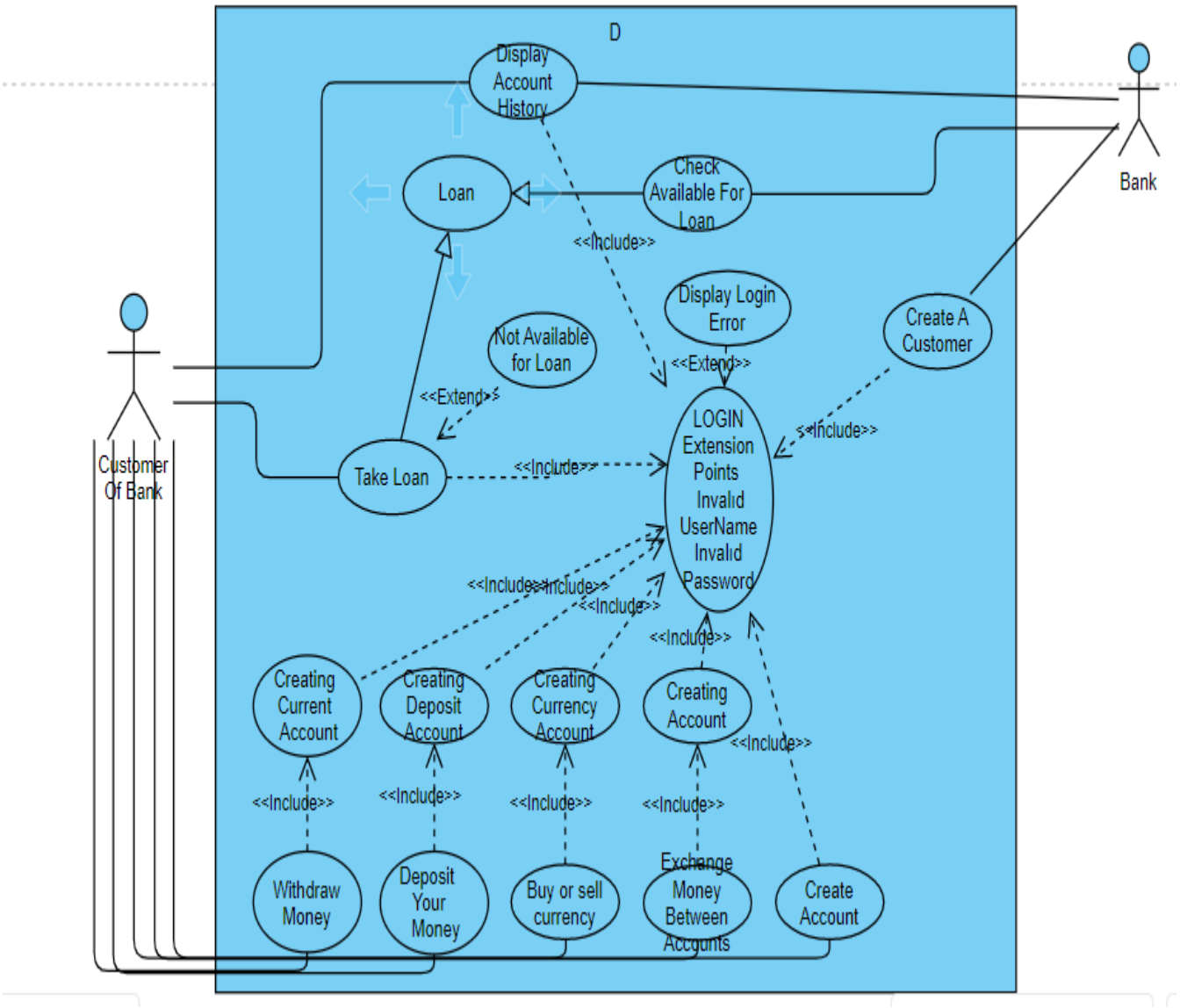
- private JButton loginButton
- private JButton resetButton
- private JButton signupButton
- private JPasswordField userPasswordField
- private JLabel messageLabel
- HashMap<String, Customer> loginInfo

Functions in this class:

- LoginPage(HashMap<String, Customer> loginInfoOriginal)
- public void actionPerformed(ActionEvent e)

CHAPTER THREE

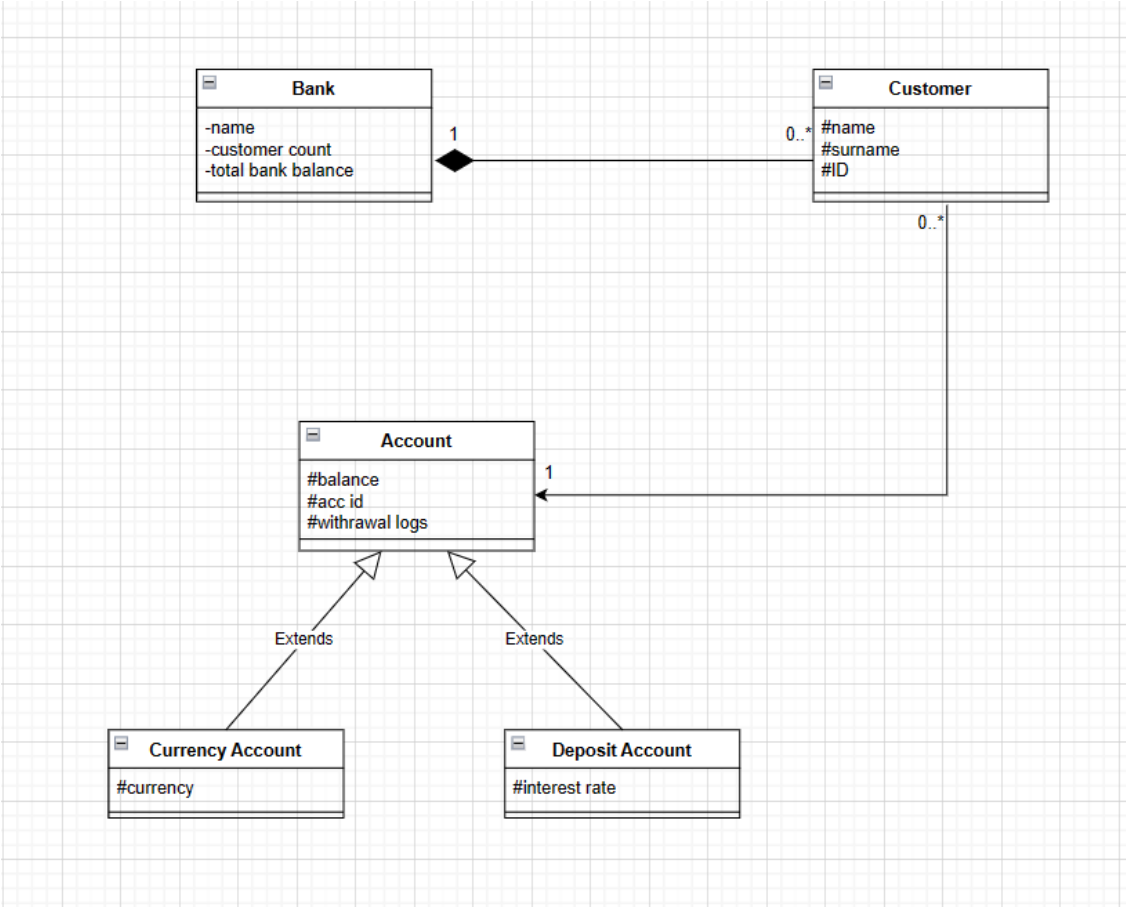
USE-CASE DIAGRAM



The use case diagram for our bank application is as seen above. The customer can only withdraw money from the current account, so a current account must be available to withdraw money. In order for any customer to create a current account, they must be logged into our bank. Every customer logged into our bank can view their account history. If a customer has only logged into our bank and created a time deposit account, he can deposit

his money at maturity. If our customers can perform transactions such as selling and buying currencies, they must log in to our bank and then create a currency account. Our bank system in the application we will make offers opportunities such as loan withdrawal to everyone who has logged into our bank apart from account transactions. Our bank representative can view their account history at any time. The general outline of our bank application that we will start to create is like this in the current part of our project.

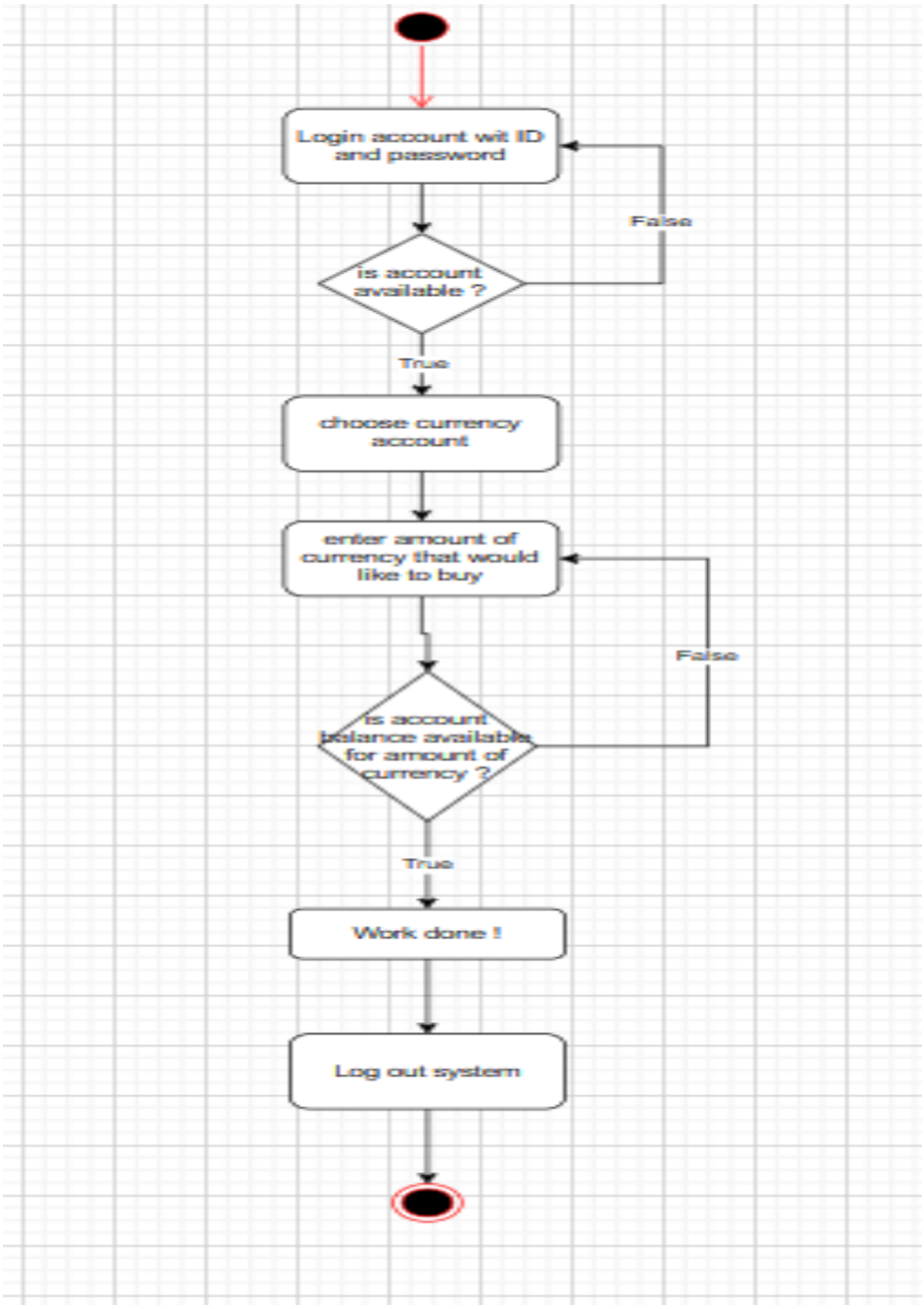
CLASS DIAGRAM



This is a class diagram. There is a main bank class that has to be. Customer connects with bank and if bank wouldnt exist customer also cannot. Every customer can open more than one account for example customers can have a deposit account and exchange account in the

same time. But every account has only one customer and also deposit and exchange accounts are subclass of account class.

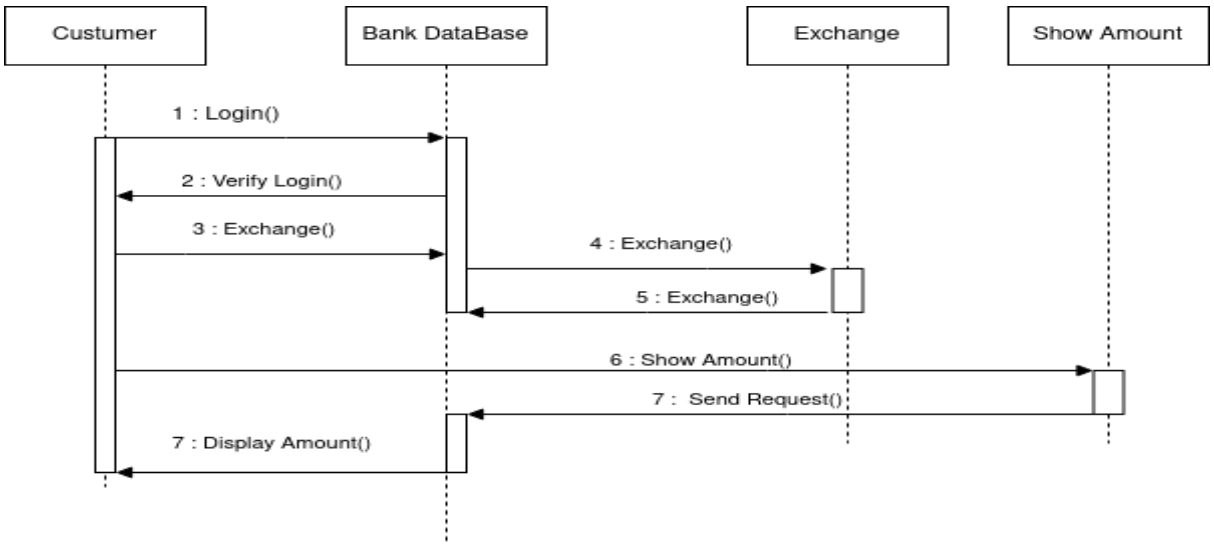
ACTIVITY DIAGRAM



This is an activity digaram about aa excanging money statement. First of all user have to login successully otherwise login get denied then user choose which currency unit wants to

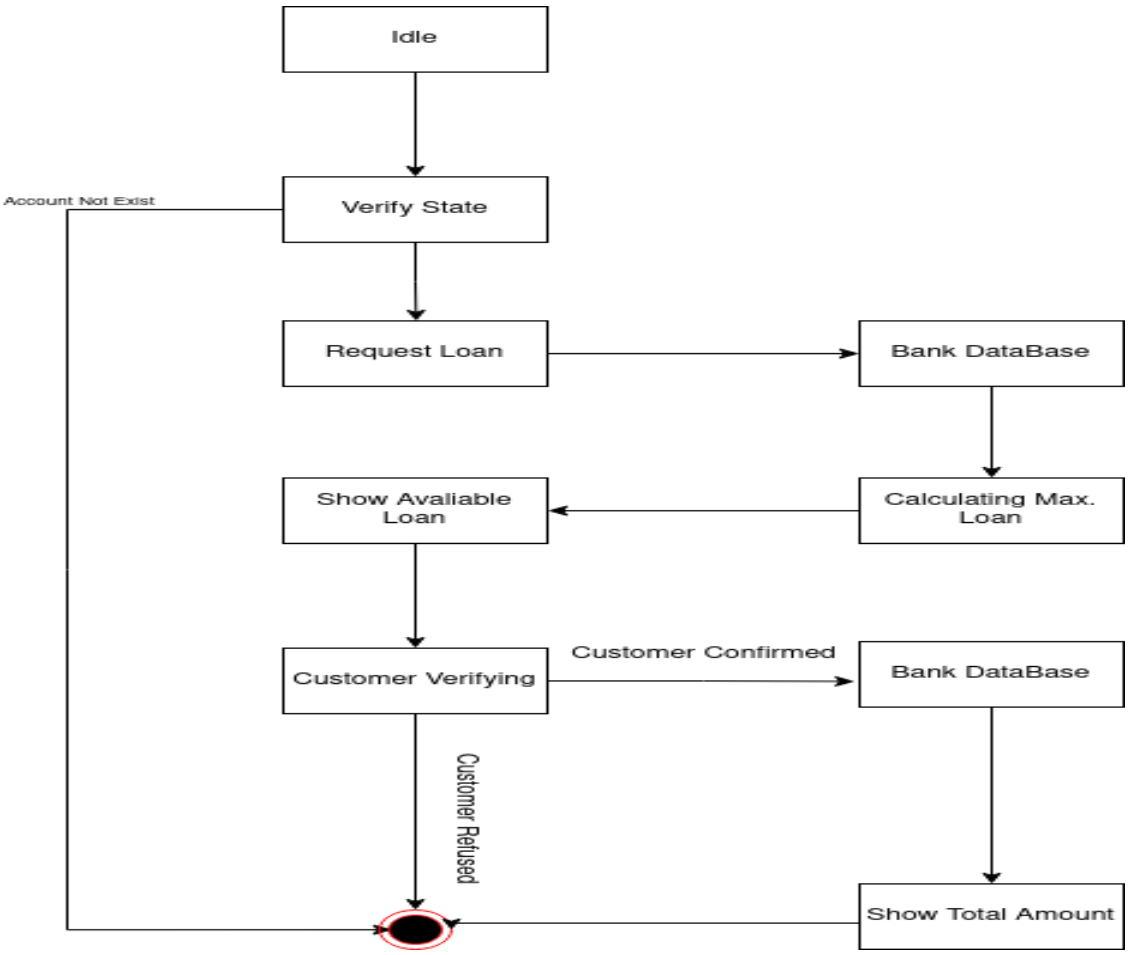
exchange. After all that situations system checks account balance if it is enough work is done if it isn't system wants to a valid number for exchange.

SEQUANCE DIAGRAM



This is a sequence diagram that shows us simple exchanging process. Firstly, customer have to log in the banking system. If customer’s info is true, customer carry out exchanging process. The main partition of this process is checking bank database. So. Customer's amounts detected and exchanging. Customer can see his/her total amount after all the process.

STATE DIAGRAM



This is a state diagram that shows us simple loan process. Firstly, customer have to log in the banking system. If customer's info is true, customer carry out loan process. The main partition of this process is calculating maximum loan for he/she. Credibility note is calculated. Customer may request max loan as algorithm calculated. If customer will confirm calculated maximum loan that will allocate his/her account. If do not all the process end and customer return main menu.

CHAPTER FOUR

CHAPTER FOUR

In this code, we see a class named "Account." Here's a detailed description of this class:

Class: Account

Variables:

accountId: A String representing the account ID.

balance: A double value representing the account balance.

logs: An ArrayList that stores the daily logs related to the account.

Methods:

Account(String accountId, double balance): The constructor method of the Account class. It takes the account ID and balance information as parameters and assigns them to the respective variables. It also creates a new ArrayList for the "logs" variable.

getAccountId(): A getter method that returns the account ID.

getBalance(): A getter method that returns the account balance.

setBalance(double balance): A setter method that sets the account balance.

getLogs(): A getter method that returns the ArrayList containing the logs related to the account.

depositMoney(double amount): A method that performs a deposit operation with the specified amount. It retrieves the current balance, adds the specified amount to it, and updates the result.

withdraw(double amount): A method that performs a withdrawal operation with the specified amount. It retrieves the current balance, subtracts the specified amount from it, and updates the result. If the balance is less than the requested amount, it prints the message "not enough money" to the screen.

This class represents a model for bank accounts. It holds information such as the account ID, balance, and daily logs. It also includes methods to perform basic operations like depositing and withdrawing money.

Class: CurrencyAccount

The "CurrencyAccount" class is derived (inherits) from the "Account" class and represents currency accounts. It has the following detailed description:

Variables:

currencyUnit: A String representing the currency unit of the account.

EURO, DOLLAR, STERLIN: Constant double values representing the conversion rates.

Methods:

CurrencyAccount(String accountId, double balance, String currencyUnit): The constructor method of the CurrencyAccount class. It takes the account ID, balance, and currency unit information as parameters and calls the constructor method of the Account class. It also assigns the value to the currencyUnit variable.

getCurrencyUnit(): A getter method that returns the currency unit of the account.

depositMoney(double amount): A method that performs a deposit operation with the specified amount. It retrieves the current balance and adds the specified amount to the balance after converting it to the account currency. The conversion calculations depend on the EURO, DOLLAR, and STERLIN values.

getEURO(), getDOLLAR(), getSTERLIN(): Getter methods that return the conversion rates.

showBalance(): A method that returns the account balance, along with the currency unit, as a text. The currency unit is appended to the balance and represented with the appropriate symbol.

This class is developed for currency accounts, inheriting the properties of the "Account" class. It can be used for special operations related to the account currency, deposit transactions, and balance display.

Class: Customer

Variables:

id: A String representing the customer's ID.
password: A String representing the customer's password.
accounts: An array of type Account that holds the customer's accounts.

Methods:

Customer(String id, String password): The constructor method of the Customer class. It takes the customer's ID and password as parameters and assigns them to the respective variables. It also creates an array of size 3 as the accounts array.

transferMoney(Customer receiver, double amount): A method that performs a money transfer operation of the specified amount to the receiving customer. It checks if both the sender's and receiver's accounts exist. If the sender's account has sufficient balance, the specified amount is deducted from the sender's account and added to the receiver's account. It returns 2 if the transfer is successful, 1 if either one or both of the accounts do not exist, and 3 if the sender's account does not have sufficient balance.

sellCurrency(double amount): A method that performs the sale of a specified amount from the customer's currency account. It first checks if the currency account has sufficient balance. Conversion calculations are performed based on the currency account's currency unit, and the resulting amount is added to the customer's general account. It returns true if the sale is successful, and false if there is insufficient balance.

createSavingAccount(): A method used to create a savings account. After the savings account is created, it is added to the accounts array.

createAccount(): A method used to create an account. After the account is created, it is added to the accounts array.

createCurrency(String currencyType): A method used to create a currency account with a specific currency type. After the currency account is created, it is added to the accounts array.

getId(): A getter method that returns the customer's ID.

setId(String id): A setter method that sets the customer's ID.

getPassword(): A getter method that returns the customer's password.

setPassword(String password): A setter method that sets the customer's password.

This class is used to represent customers and manage their accounts and transactions. It allows for managing accounts, creating new accounts, performing money transfers, and currency sales.

Class: IdAndPasswords

In this code, there is a class named "IdAndPasswords". This class manages a HashMap, which is a data structure that stores customer IDs and passwords. Here is a detailed description of this class:

Variables:

ids: A String array that stores customer IDs.

passwords: A String array that stores customer passwords.

a: A Customer array that stores Customer objects.

loginInfo: A HashMap that stores customer objects associated with their IDs.

Methods:

IdAndPasswords(): The constructor method of the IdAndPasswords class. It creates arrays containing customer IDs and passwords. Then, it creates Customer objects with these IDs and passwords, and adds these objects to the loginInfo HashMap.

getLoginInfo(): A getter method that returns the loginInfo HashMap.

This class is used to store and verify customer IDs and passwords. You can use this class when you need to match customer login information.

Class: LoginPage

In this code, a login page has been designed. It creates a user interface that includes username and password fields. Here is a detailed description of this class:

Variables:

frame: The main JFrame of the login page.

loginButton: The login button.

resetButton: The reset button.

signupButton: The signup button.

userIDField: The username field (JTextField).

userPasswordField: The password field (JPasswordField).

userIDLabel: The username label (JLabel).

userPasswordLabel: The password label (JLabel).

messageLabel: The label that displays operation results (JLabel).

loginInfo: A HashMap that contains user login information.

Methods:

LoginPage(HashMap<String, Customer> loginInfoOriginal): The constructor method of the LoginPage class. It creates the login page and adds the necessary components. It also takes the HashMap that stores login information as input.

actionPerformed(ActionEvent e): The actionPerformed method that comes from the ActionListener interface. It handles the button click events. When the reset button is clicked, it clears the username and password fields. When the login button is clicked, it verifies the username and password, and updates the message accordingly. When the signup button is clicked, it opens the signup page and retrieves the new login information.

This class is used to design a login page and manage the login process. It verifies user credentials and, if the login is successful, it calls the "WelcomePage" class. Additionally, it uses the "signUpPage" class for signup operations.

Class: signUpPage

This code represents a signup page. It creates a user interface that includes username, password, and signup button. Here is a detailed description of this class:

Variables:

frame: The main JFrame of the signup page.

resetButton: The reset button.

signupButton: The signup button.

userIDField: The username field (JTextField).

userPasswordField: The password field (JPasswordField).

userIDLabel: The username label (JLabel).

userPasswordLabel: The password label (JLabel).

messageLabel: The label that displays operation results (JLabel).

loginInfo: A HashMap that contains user login information.

Methods:

signUpPage(HashMap<String, Customer> loginInfoOriginal): The constructor method of the signUpPage class. It creates the signup page and adds the necessary components. It also takes the HashMap that stores login information as input.

actionPerformed(ActionEvent e): The actionPerformed method that comes from the ActionListener interface. It handles the button click events. When the signup button is clicked, it retrieves the username and password, creates a new Customer object, and adds this information to the loginInfo HashMap. As a result, it updates a message and closes the page. When the reset button is clicked, it clears the username and password fields.

This class is used to design a signup page and manage the signup process. It allows the user to create a new account and adds the login information to the loginInfo HashMap.

Class: SavingAccount

the class SavingAccount is derived from the Account class. This means that the SavingAccount class inherits the properties and behaviors of the Account class. SavingAccount can be considered as a subset of Account, and it can utilize the properties and methods of the Account class.

Since SavingAccount inherits all the properties and behaviors of the Account class, it can access and use basic attributes such as accountId and balance. Additionally, it can directly use the methods defined in the Account class (e.g., getAccountId, getBalance, withdrawMoney) or override them to add its own unique behaviors if needed.

This inheritance relationship indicates that the SavingAccount class is a type of Account. It can be seen as an extended version or a specialized subclass of the Account class. By doing so, it improves the programming logic and code organization by avoiding code duplication and enabling reuse of existing code.

Variables:

interestRate: A private double variable that holds the interest rate. It is initialized with a default value of 0.1.

Methods:

SavingAccount(String accountId, double balance): A constructor method that takes accountId and initial balance values. It calls the constructor of the superclass Account using the super() keyword to set the accountId and balance values. Additionally, the interestRate variable is set to 0.1.

getinterestRate(): A method that returns the interest rate.

depositMoney(int amount): A method that deposits a specified amount of money into the account. To perform the operation, the current account balance (balance) is temporarily stored in a variable (balance) and then added to the specified amount (amount) to obtain the new account balance (this.balance).

applyInterest(): A method that applies interest to the account. To perform the operation, the current account balance (balance) is temporarily stored in a variable (balance). Then, the interest amount (newBalance) is calculated by multiplying the balance by the interest rate (interestRate). This interest amount is added to the current account balance (balance) to obtain the new account balance (this.balance).

This class represents a savings account and uses the interestRate variable to store the interest rate. Additionally, it defines the depositMoney method to deposit money into the account and the applyInterest method to apply interest to the account.

that represents a graphical user interface (GUI) for a banking application. It provides various functionality related to account management, transactions, and balance inquiries. Here is a detailed explanation of the code:

1. The code imports necessary Java libraries for GUI components, event handling, and data structures:
    - `import java.awt.Color;`: Imports the Color class for specifying colors in GUI.
    - `import java.awt.Dimension;`: Imports the Dimension class to set the dimensions of GUI components.
    - `import java.awt.FlowLayout;`: Imports the FlowLayout class for arranging GUI components.
    - `import java.awt.event.ActionEvent;` and `import java.awt.event.ActionListener;`: Imports classes for handling GUI events.
    - `import java.util.HashMap;`: Imports the HashMap class for storing customer information.
    - `import javax.swing.*;`: Imports various classes from the Swing library for creating the GUI.
2. The code defines a class named `WelcomePage` that extends the `JFrame` class and implements the `ActionListener` interface. This means that the `WelcomePage` class represents a frame window and can handle GUI events.
3. The code declares and initializes several instance variables that represent GUI components such as buttons, labels, text fields, radio buttons, panels, button groups, etc. These variables are used to create and interact with the GUI elements.
4. The `WelcomePage` class constructor takes two parameters: a `HashMap<String, Customer>` named `loginInfoOriginal` and a `String` named `idOfCustomer`. These parameters are used to initialize the state of the `WelcomePage` object. The constructor sets up the frame's properties, sets the title, size, location, layout, and visibility. It also calls two methods: `setupInputs()` and `setupCommands()`, which create and configure the input and command panels of the GUI.
5. The `setupInputs()` method is responsible for creating and configuring the input panel of the GUI. It creates various GUI components such as labels, text fields, radio buttons, and button groups and adds them to the input panel. It also sets up event listeners for the buttons and radio buttons.
6. The `setupCommands()` method creates and configures the command panel of the GUI. It creates buttons for various banking operations such as account creation, balance inquiry, withdrawal, deposit, interest calculation, interest application, and money transfer. It also sets up event listeners for the buttons.
7. The `actionPerformed()` method is implemented as part of the `ActionListener` interface and handles events generated by GUI components. It checks the source of the event and performs different operations based on the button clicked. The operations include creating an account, checking balance, depositing money, calculating interest, transferring money, and exiting the application.

8. The `showInterest()` method displays the interest rate for a saving account if it exists.
9. The `deposit()` method handles the deposit operation. It checks the selected account type (account, saving, or currency) and performs the deposit operation accordingly.
10. The `checkBalance()` method retrieves the current balance of the selected account and updates the balance text field accordingly.
11. The `checkAccount()` method checks if the selected account type exists for the current customer.
12. The `createAccount()` method creates a new account based on the selected account type.
13. The `transferMoney()` method handles the money transfer operation between two accounts. It checks if the selected account type is an account and if the receiver's account exists. It then performs the transfer operation.
14. The `applyInterest()` method applies interest to the saving account if it exists.
15. The `withdraw()` method handles the withdrawal operation. It checks the selected account.

CHAPTER FIVE

CONCLUSION AND FUTURE WORKS

This project consists of a Java program that represents the graphical user interface (GUI) of a banking application. The program provides various functionalities such as account

management, transactions, and balance inquiries. It uses a HashMap to store customer information.

The code defines variables to create and interact with GUI components. Then, it defines a class named "WelcomePage" that extends the JFrame class and implements the ActionListener interface. This class represents a window with the ability to handle GUI events.

This project provides a Java program for performing basic banking operations. However, to make the project more professional in the future, some additions can be made:

1. Database integration: Storing customer information and account data in a database enhances security and provides a more scalable solution.

2. Security measures: Stronger security measures should be added for user login and transaction authentication. For example, password complexity requirements, session timeouts, secure communication (HTTPS), etc.

3. More comprehensive error handling: The program should effectively handle user errors and report issues arising from incorrect inputs with user-friendly messages.

4. Expansion of functionality: In addition to banking operations, the program could consider providing more functionalities such as bill payments, credit applications, account summaries, etc.

5. Interface improvements: User-friendly interface design and layout enhancements give the program a more professional and appealing look.

6. Documentation: Comprehensive documentation should be prepared regarding the development, usage, and configuration of the project. This facilitates better understanding and enables others to contribute to the project.

These additions will make the project more secure, user-friendly, and functional.