OBJECT ORIENTED PROGRAMMING

FALL 2023_CSE 2103 SECTION 2

SUBMITTED BY GROUP 1

Md. Selim Ahmed [223014014]

SUBMITTED TO

Submitted to: Ferdous Bin Hafiz

Submission Date: 25/11/2023

UNIVERSITY OF LIBERAL ARTS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OPEN ENDED: A JAVA PROJECT FOR A BANK MANAGEMENT APPLICATION

TITLE

Creating a java based program based bank management system.

OBJECTIVES

- Demonstrating the fundamental concepts of object-oriented programming like encapsulation, polymorphism, inheritance and abstraction.
- Applying the concepts of object-oriented programming for solving real-world problems
- Applying the java Graphical user interface for taking input also the output.
- Various method, class, Jframe for implement the system.

INTRODUCTION

This report showcases the implementation of Java's object-oriented programming principles through a comprehensive bank management platform, demonstrating encapsulation, inheritance, polymorphism, and other vital aspects of OOP. The bank management system offers a range of options for clients, beginning with a login panel that comprises separate sections for administrators and clients.

Upon creating an account, clients gain access to a user-friendly login and signup interface. Once logged in, clients have the privilege to review their account details, execute deposits, withdrawals, and apply for loans. Additionally, they can seamlessly track their transaction history, with all updated information conveniently displayed within their user profile. This streamlined system ensures a hassle-free banking experience for clients, enabling them to manage their finances efficiently.

Administrators, upon accessing the system, are equipped with comprehensive control over client information. They can effortlessly view a table containing vital client details and exercise administrative functionalities such as deletion, updating records, and managing loan requests. The pivotal role of administrators lies in approving or declining loan applications, especially in scenarios where multiple requests are made. All critical data is securely stored in files, ensuring confidentiality and reliability within the system.

This platform effectively encapsulates the functionalities required for both clients and administrators, ensuring efficient management and secure storage of crucial banking information.

PROBLEM UNDERSTANDING

DESIGN OVERVIEW:

The objective is to create a bank management application capable of handling multiple users and their performance fields while ensuring data integrity and security through encapsulation. Utilizing inheritance, subclasses will be created from superclasses to establish various account types and operations, all inheriting base templates. Java Swing will be employed for designing the bank management system's interface.

FUNCTIONALITIES OVERVIEW:

1. **Login Interface (JFrame: "Login"):**
    - Input fields and buttons for user credentials.

- Method creation for the login panel.
- Utilization of BufferWriter and BufferReader to store and retrieve data.

2. **Client Operations:**
   - SignUp function through a JFrame for inputting necessary information.
   - Validation to prompt completion of all required fields.
   - Adding client information to the table via an "addClient" method within the client ArrayList.
   - Creating accounts, handling deposits, loans, and displaying updated information.
3. **Interface Navigation:**
   - Transition between JFrames by setting visibility (false to true).
4. **Account Operations:**
   - Functionality for depositing funds via a dedicated JFrame and storing the input in an ArrayList.
   - Displaying updated information through a "display" method.
   - Withdrawal functionality within the user interface JFrame.
   - Transaction management via a separate class with getters, setters, and relevant methods in the JFrame.
5. **Loan Handling:**
   - Method implementation for loan management, including setting maximum limits and prompting confirmation for loan requests exceeding limits.
6. **Admin Profile:**
   - Functionality post-admin login to display client information.
   - Methods for deleting clients, updating records, accepting/declining loan requests.
7. **File Handling:**
   - FileManager class creation for writing and reading functions.
   - Writing data to a file upon function closure and reading information at project runtime, storing it in an array.

BACKGROUND THEORY

POLYMORPHISM

Polymorphism in Java is a key principle in object-oriented programming that allows objects of different classes to be treated as objects of a common superclass through a shared interface. It enables a single interface to be used for a general class of actions.

- **Method Overloading:** One form of polymorphism in Java is method overloading. It occurs when a class has multiple methods with the same name but different parameters. This allows the methods to perform similar tasks with different inputs.
- **Method Overriding:** Inheritance is another aspect of polymorphism. When a subclass inherits from a superclass, it can override methods defined in the superclass. This means that a method in a subclass has the same name, return type, and parameters as a method in its superclass, but provides different functionality.
- **Interfaces:** Java interfaces provide a way to achieve polymorphism by allowing classes to implement the same interface but provide their own implementation for the methods declared in the interface. This allows different classes to be treated uniformly via their common interface.
- **Dynamic Binding:** Polymorphism in Java involves dynamic method binding, where the method to be executed is determined during runtime based on the object being referred to by a reference variable. This allows flexibility in invoking methods, enabling code to be more adaptable to different object types.

- **Generics:** Java's generics feature also contributes to polymorphism by allowing classes and methods to operate on objects of various types parameterized during their declaration. This helps in writing more reusable and flexible code.

Overall, polymorphism in Java promotes flexibility, reusability, and extensibility in code by allowing different objects to be treated uniformly through a common interface or superclass, enhancing the overall structure and maintainability of programs.

INHERITANCE

Inheritance in Java is a fundamental concept in object-oriented programming. It allows you to create new classes based on existing ones, enabling the reuse of methods and fields from the parent class.

Here are some key points about inheritance in Java:

- Inheritance creates a hierarchy between classes, where a child class (also known as a subclass) inherits the properties and behaviors of a parent class (also known as a superclass).
- The child class can access and use the methods and fields of the parent class, as well as define its own additional methods and fields.
- Inheritance promotes code reuse and modularity, as common attributes and behaviors can be defined in a parent class and inherited by multiple child classes.
- In Java, the "extends" keyword is used to establish an inheritance relationship between classes. For example, if class B extends class A, then class B is the child class and class A is the parent class.
- Inherited methods and fields can be overridden in the child class to provide a different implementation or behavior. This allows for customization and specialization of inherited functionality.
- Java supports single inheritance, which means that a class can only inherit from one parent class. However, multiple levels of inheritance are possible, where a child class can become a parent class for another child class.
- Inheritance is an important part of the object-oriented programming paradigm, as it facilitates code organization, promotes code reuse, and allows for the creation of hierarchies of related classes.

Overall, inheritance in Java provides a powerful mechanism for creating class hierarchies and reusing code. It allows for the creation of more specialized classes that inherit and extend the functionality of existing classes, leading to more efficient and modular code.

ABSTRACT CLASS

An abstract class in Java is a class that is declared with the "abstract" keyword. It serves as a blueprint for other classes and cannot be instantiated on its own. Instead, it must be inherited by other classes.

Here are some key points about abstract classes in Java:

- An abstract class can have both abstract and non-abstract methods. Abstract methods are declared without a body and must be implemented by the subclasses that inherit from the abstract class.
- Abstract classes can also have non-abstract methods with a body, which can be directly used by the subclasses.

- Abstract classes are useful when you want to define common functionality that should be shared among multiple subclasses.
- Abstract classes can provide a level of abstraction and serve as a template for other classes.
- Abstract classes cannot be instantiated, meaning you cannot create objects of an abstract class directly.
- To use an abstract class, you need to create a subclass that extends the abstract class and provide implementations for all the abstract methods.

Overall, abstract classes in Java provide a way to define common behavior and enforce certain methods to be implemented by the subclasses, allowing for better code organization and reusability.

ARRAY LIST:

An ArrayList in Java is a class that is part of the Java collection framework. It provides a dynamic array implementation, allowing you to store and manipulate elements in a flexible manner.

Here are some key points about ArrayList in Java:

- **Resizable array:** The ArrayList class is essentially a resizable array. It can grow or shrink dynamically based on the number of elements it contains. This makes it convenient for situations where you need to add or remove elements frequently.
- **Java collection framework:** ArrayList is part of the Java collection framework, which provides a set of classes and interfaces for handling collections of objects. It is located in the java.util package.
- Adding and removing elements: You can add elements to an ArrayList using the add() method, and remove elements using the remove() method. The ArrayList automatically adjusts its size to accommodate the changes.
- **Random access:** ArrayList allows for efficient random access to elements. You can retrieve elements by their index using the get() method. This makes it easy to access and modify specific elements within the ArrayList.
- **Iterating over elements:** You can iterate over the elements of an ArrayList using a loop or the enhanced for loop. The size() method can be used to determine the number of elements in the ArrayList.
- **Dynamic size:** Unlike regular arrays, ArrayLists do not have a fixed size limit. You can add or remove elements at any time without worrying about exceeding the size limit.
- **Generic type:** ArrayLists in Java can be parameterized with a specific type. This allows you to create ArrayLists that can only store elements of a certain type, ensuring type safety.
- **Performance considerations:** While ArrayLists provide flexibility and convenience, they may not be the most efficient choice for certain operations. For example, inserting or removing elements in the middle of an ArrayList can be slower compared to other data structures like LinkedList.

Overall, ArrayLists in Java are a versatile and widely used data structure that provides dynamic array functionality. They are commonly used when you need a resizable array that supports random access and efficient element manipulation.

SCANNER

The Scanner class in Java is a utility class that allows you to read input from different sources, such as the console or a file. It is part of the java.util package and provides methods to parse and process various types of data.

Here are some key points about the Scanner class:

- **Import:** To use the Scanner class, you need to import it using the following statement: import java.util.Scanner;
- **Creating an instance:** To create an instance of the Scanner class, you can use the following syntax: Scanner scanner = new Scanner(System.in); This creates a Scanner object that reads input from the standard input, which is usually the keyboard.
- **Reading input:** The Scanner class provides different methods to read input of various types, such as integers, doubles, strings, and more. For example, you can use nextInt() to read an integer, nextDouble() to read a double, and nextLine() to read a whole line of text.
- **Validating input:** The Scanner class also provides methods to check if there is more input available or if the next input matches a specific data type. These methods, such as hasNextInt() or hasNextDouble(), can be used to validate user input before reading it.
- **Exception handling:** It is important to handle exceptions when using the Scanner class. For example, it can throw an InputMismatchException if the input does not match the expected data type. Proper exception handling ensures that your program handles unexpected input gracefully.
- **Closing the Scanner:** After reading the input, it is good practice to close the Scanner object to release system resources. You can do this by calling the close() method on the Scanner object, like this: scanner.close()
- **File input:** In addition to reading input from the console, the Scanner class can also be used to read input from files. Instead of passing System.in as a parameter when creating the Scanner instance, you can pass a File object representing the file you want to read from.

The Scanner class is a useful tool for reading input in Java, whether it is from the console or a file. It simplifies the process of interacting with users and parsing different types of data.

ENCAPSULATION

Encapsulation in Java is a fundamental concept in object-oriented programming that involves hiding the internal implementation details of a class and exposing only a public interface. It allows for the bundling of data and methods within a single unit, similar to a capsule containing multiple medicines.

Here are some key points about encapsulation in Java:

- **Data hiding:** Encapsulation enables the hiding of internal data members of a class, preventing direct access from outside. This helps in maintaining data integrity and security.
- **Access modifiers:** Java provides access modifiers such as public, private, and protected to control the visibility of class members. By using private access modifiers, we can restrict direct access to the internal state of a class.
- **Getters and setters:** Encapsulation encourage the use of getter and setter methods to access and modify the private data members of a class. Getters are used to retrieve the values of private variables, while setters are used to set or modify the values.
- **Information hiding:** Encapsulation allows the class to control how its data is accessed and modified. It provides a level of abstraction, where the internal implementation details are hidden and only the necessary information is exposed.
- **Code maintainability:** Encapsulation promotes code maintainability by encapsulating related data and methods within a class. This makes it easier to understand, modify, and extend the code without affecting other parts of the program.

- **Code reusability:** Encapsulation facilitates code reusability by creating reusable classes with well-defined interfaces. These classes can be used in different parts of the program without exposing their internal implementation.
- **Encapsulation and inheritance:** Encapsulation and inheritance are closely related concepts in object-oriented programming. Encapsulation allows for the encapsulation of data and methods within a class, while inheritance enables the reuse of code and the extension of existing classes.

In summary, encapsulation in Java is a powerful mechanism that promotes data hiding, code maintainability, and code reusability. It allows for the creation of well-structured and modular programs by encapsulating related data and methods within classes and controlling their access through well-defined interfaces.

FILEREADER

- **Purpose:** FileReader reads character files in a platform-independent manner by reading bytes and decoding them into characters using the default character encoding.
- **Usage:** It is used for reading streams of characters from a file.
- **Character Encoding:** Reads characters using the system's default character encoding unless specified otherwise.
- **Read Operations:** Provides methods like read(),read(char[] cbuf, int off, int len), etc., for reading characters or arrays from a file.

FILEWRITER

- **Purpose:** FileWriter writes character files in a platform-independent manner by writing bytes encoded from characters using the default character encoding.
- **Usage:** It is used for writing streams of characters to a file.
- **Character Encoding:** Writes characters using the system's default character encoding unless specified otherwise.
- **Write Operations:** Offers methods like write(String str), write(char[] cbuf, int off, int len), etc., for writing characters or arrays to a file.

BUFFEREDREADER

- **Purpose:** BufferedReader reads text from a character-input stream, buffering characters to provide efficient reading of characters, arrays, or lines.
- **Buffering**: It buffers the input, allowing for the reading of characters, lines, or arrays from the underlying stream in a more efficient manner compared to directly reading from the stream.
- **Read Operations:** Offers methods like read(),readLine(), and read(char[] cbuf, int off, int len) for reading characters or lines from the input stream.

BUFFEREDWRITER

- **Purpose:** BufferedWriter writes text to a character-output stream, buffering characters to provide efficient writing of characters, arrays, or strings.
- **Buffering:** It buffers the output, allowing for efficient writing of characters, arrays, or strings to the underlying stream.

- **Write Operations:** Provides methods like write(String str), write(char[] cbuf, int off, int len), and newLine()for writing characters, arrays, or strings to the output stream. The newLine()method writes a platform-specific line separator.

ALGORITHM DESIGN

PROFILE CLASS:

1. **Attributes:**
   - username, email, password, gender (private)
   - Profiles (static ArrayList of Profiles)
2. **Constructor:**
   - Initializes username, email, password, gender.
   - Adds the current instance to Profiles.
3. **Getter and Setter Methods:**
   - attributes (username, email, password, gender).

CLIENT CLASS (INHERITS PROFILE):

1. **Attributes:**
   - balance, loan, requestedLoan (private)
   - Transactions (ArrayList of Transaction objects)
2. **Constructor:**
   - Inherits from Profile and sets balance, loan, requestedLoan.
   - Initializes Transactions ArrayList.
3. **Getter and Setter Methods:**
   - For balance, loan, requestedLoan.
   - getTransactions()for the Transactions ArrayList.
4. **Transaction Methods:**
   - deposit(int Amount)
   - withdraw(int Amount)
   - borrow(int Amount)
   - repayLoan(int Amount)
   - addTransaction(Transaction T)

ADMIN CLASS (INHERITS PROFILE):

1. **Attributes:**
   - LoanRequests (static ArrayList of Client objects)
2. **Constructor:**
   - Inherits from Profile.
3. **Getter Method:**
   - getLoanRequests()for the LoanRequests ArrayList.
4. **Admin Operations:**
   - acceptLoan(Client p)
   - rejectLoan(Client p)

- removeProfile(Client p)

TRANSACTION CLASS:

1. **Attributes:**
   - Amount, Type (private)
2. **Constructor:**
   - Initializes Amount, Type.
3. **Getter and Setter Methods:**
   - For Amount, Type.

FILEMANAGER CLASS:

Methods:

1. **writeProfilesToFile()**
   - Opens a file writer.
   - Iterates through Profiles.
   - Writes Client/Admin details and transactions to the file.
2. **readProfilesFromFile()**
   - Opens a file reader.
   - Reads each line to parse and create Client/Admin objects.
   - Populates profiles and updates Profile.Profiles.
   - Handles LoanRequests for Admins.

MAIN PROGRAM EXECUTION:

INITIALIZATION AND UI MANAGEMENT:

Initialize UI on Startup:

- Set initial visibility of different panels (LoginPanel, SignUpPanel, ClientPanel, AdminPanel).

Main Method:

- Create an instance of the UserInterface.
- Set it visible and initialize the UI startup.

CLIENT INTERFACE HANDLING:

updateClientValues() Method:

- Update client-specific UI components based on the logged-in profile's data:
- Set client names, balance, loan, and recent transaction details.
- Update specific UI components (labels, fields) with corresponding client data.

Client Interaction Methods (Deposit, Withdraw, Loan):

- Validate input fields.
- Perform the deposit, withdrawal, and loan operations for the logged-in client.
- Update UI after each operation.

ADMIN INTERFACE HANDLING:

updateAdminValues() Method:

- Update admin-specific UI components with the logged-in admin's data.

Admin Requests Handling (Accept, Reject):

- Process pending loan requests - accept or reject based on admin actions.
- Update UI accordingly.

USER INTERACTION HANDLERS:

Event Handlers:

- Actions triggered on button clicks, text field entries, and focus changes.
- Includes validation checks and UI updates.
- Transition between panels (LoginPanel, SignUpPanel, ClientPanel, AdminPanel) based on user actions (e.g., log in, sign up, log out).

DATA MANAGEMENT:

Profile and Login Handling:

- Login using email and password.
- Check for existing profiles, validate input, handle login errors, and update the logged-in profile.

Sign-Up Handling:

- Validate sign-up inputs, check for existing profiles, create a new profile, and update UI accordingly.

UI COMPONENTS:

Field Handlers:

- Manage focus events and text input in various text fields (FirstNameSignUp, LastNameSignUp, EmailSignUp, etc.).
- Handle placeholder texts, clear fields on focus, and validate input data.

TABLE AND DATA POPULATION:

Populate Tables (AllRequestTable, AllProfileTable):

- Refresh tables with the latest data based on requests and profiles.
- Update row counts, populate rows with corresponding profile or request details.

INITIALIZATION AND UI MANAGEMENT METHODS:

Initialization Methods:

- Clear and initialize UI components or panels to their initial states.
- Handle panel visibility and transitions.

UserInterface Constructor:

It is the constructor for a Java Swing JFrame named UserInterface. Let's break down its functionality:

1. **initComponents():** This method initializes all components present in the JFrame. It's autogenerated code created by a GUI builder tool or IDE that sets up the layout and components of the UI.
2. **FileManager.readProfilesFromFile():** This line reads profiles from a file using the FileManager class. It populates the application with existing profiles stored in a file when the application starts.
3. **addWindowListener():** Attaches a WindowListener to the JFrame. Specifically, it adds a WindowAdapter to handle the window-closing event.
   - windowClosing(WindowEvent e): Overrides the windowClosing method of the WindowAdapter to define what happens when the window is closing.
   - FileManager.writeProfilesToFile(): Writes profiles to a file using the FileManager class. This method is triggered when the window is closing, ensuring that the profiles are saved back to the file before the application terminates.

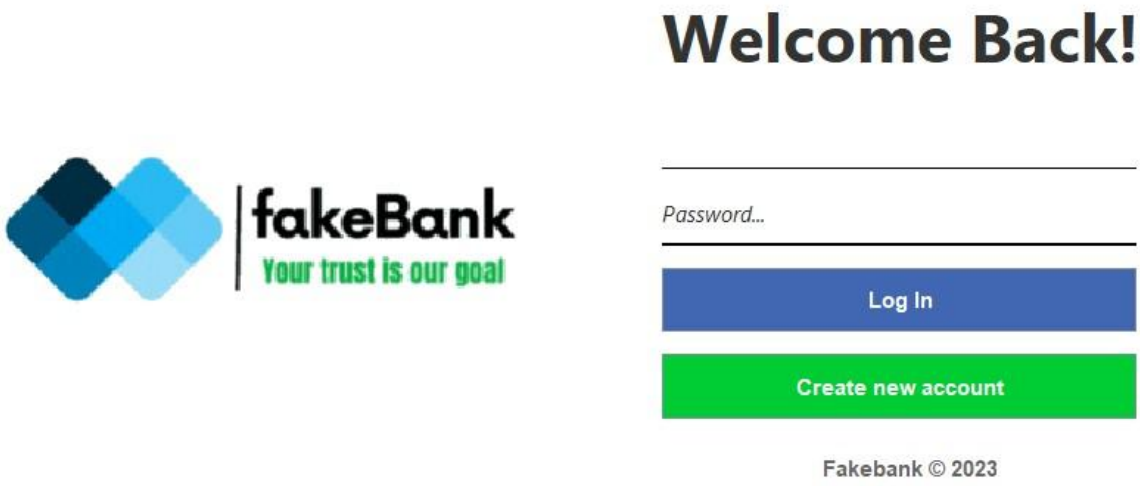
MODERN TOOLS

I have used JDK 17 as a compiler and Apache NetBeans 19 is used as IDE.

CODE

LOG IN JFRAME

- In this JFrame for taking input use the JtextField
- And for login and create new account use JButton



Gained Focus and gained lost method:

- In the Textfield box for show suggestion use gainfocus method
- When load the in the text field show the suggestion name
- In the Textfield box forremove the suggestion use gainfocuslost method
- When started type it fill blank the textField

```
private void EmailLoginFocusLost(java.awt.event.FocusEvent evt) {
    if (EmailLogin.getText().equals(anObject: "")) {
        EmailLogin.setText(t: "Email address...");
        EmailLogin.setForeground(new Color(r: 51,g: 51,b: 51));
    }
}

private void EmailLoginActionPerformed(java.awt.event.ActionEvent evt) {

}

private void PasswordLoginFocusGained(java.awt.event.FocusEvent evt) {
    if (PasswordLogin.getText().equals(anObject: "Password...")) {
        PasswordLogin.setText(t: "");
        PasswordLogin.setForeground(new Color(r: 51,g: 51,b: 51));
    }
}

private void PasswordLoginFocusLost(java.awt.event.FocusEvent evt) {
    if (PasswordLogin.getText().equals(anObject: "")) {
        PasswordLogin.setText(t: "Password...");
        PasswordLogin.setForeground(new Color(r: 51,g: 51,b: 51));
    }
}

private void PasswordLoginActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

public void clearLoginPanel(){
    EmailLogin.setText(t: "Email address...");
    PasswordLogin.setText(t: "Password...");
}
```

Log In Button:

In this check the given password and username is coorect or not if math then go to account panel if not show wrong.And admin panel go to the account list.

```
private void LoginButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (EmailLogin.getText().equals(anObject:"")){
        JOptionPane.showMessageDialog(parentComponent:this, message:"Insert email");
        return;
    }

    if (PasswordLogin.getText().equals(anObject:"")){
        JOptionPane.showMessageDialog(parentComponent:this, message:"Enter password");
        return;
    }

    for(Profile p:Profile.Profiles){
        if(p.getEmail().equals(anObject:EmailLogin.getText())){
            if(p.getPassword().equals(anObject:PasswordLogin.getText())){
                LoggedInProfile = p;
                clearLoginPanel();
                LoginPanel.setVisible(aFlag: false);
                if(p instanceof Client){
                    updateClientValues();
                    ClientPanel.setVisible(aFlag: true);
                    return;
                }else if(p instanceof Admin){
                    return;
                }else if(p instanceof Admin){
                    updateAdminValues();
                    AdminFrames.setSelectedIndex(index: 0);
                    refreshRequests();
                    AdminPanel.setVisible(aFlag: true);
                    return;
                }

            }else{
                JOptionPane.showMessageDialog(parentComponent:this, message:"Password does not mat
                return;
            }
        }
    }
    JOptionPane.showMessageDialog(parentComponent:this, message:"Profile with given email does not
}

private void GoToSignUpActionPerformed(java.awt.event.ActionEvent evt) {
    clearLoginPanel();
    LoginPanel.setVisible(aFlag: false);
    SignUpPanel.setVisible(aFlag: true);
}
```

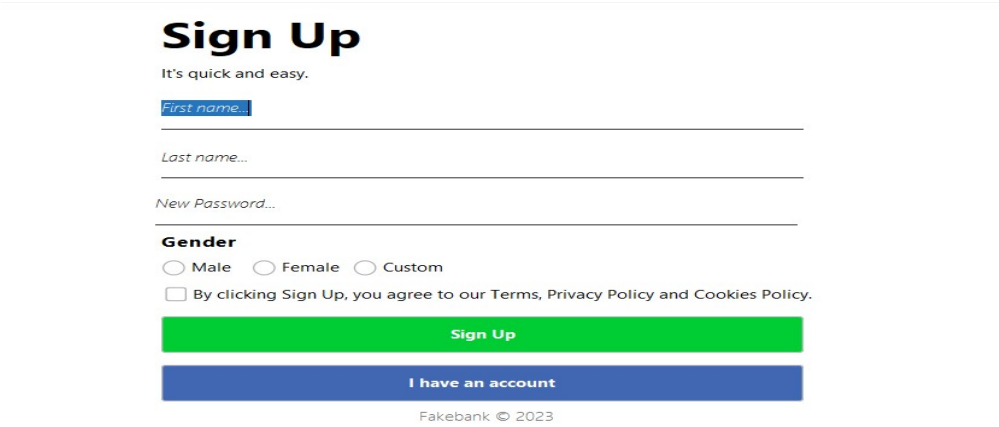
Create an Account Button:

In this button go to the sign up panel.

```
    private void GoToSignUpActionPerformed(java.awt.event.ActionEvent evt) {
        clearLoginPanel();
        LoginPanel.setVisible(aFlag: false);
        SignUpPanel.setVisible(aFlag: true);
    }
```

Sign Up jFrame

- In this JFrame firstly take Jpanel then for taking input use jTextField and for Write use JLebel
- For taking gender use JRadiobutton and choose 1 from multiple use the buttonGroup and add them in the buttonGroup.
- For term and privacy input use checkbox.
- And sign up and I have an account use the JButton.



Gained Focus method:

- In the Textfield box for show suggestion use gainfocus method
- When load the in the text field show the suggestion name

Focus Lost method:

- In the Textfield box forremove the suggestion use gainfocuslost method
- When started type it fill blank the textField

```
    private void FirstNameSignUpFocusGained(java.awt.event.FocusEvent evt) {
        if (FirstNameSignUp.getText().equals(anObject: "First name...")) {
            FirstNameSignUp.setText(t: "");
            FirstNameSignUp.setForeground(new Color(r: 0, g: 0, b: 0));
        }
    }

    private void FirstNameSignUpFocusLost(java.awt.event.FocusEvent evt) {
        if (FirstNameSignUp.getText().equals(anObject: "")) {
            FirstNameSignUp.setText(t: "First name...");
            FirstNameSignUp.setForeground(new Color(r: 0, g: 0, b: 0));
        }
    }

    private void FirstNameSignUpActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void LastNameSignUpFocusGained(java.awt.event.FocusEvent evt) {
        if (LastNameSignUp.getText().equals(anObject: "Last name...")) {
            LastNameSignUp.setText(t: "");
```

```
    private void LastNameSignUpFocusLost(java.awt.event.FocusEvent evt) {
        if (LastNameSignUp.getText().equals(anObject: "")) {
            LastNameSignUp.setText(t: "Last name...");
            LastNameSignUp.setForeground(new Color(r: 51, g: 51, b: 51));
        }           // TODO add your handling code here:
    }

    private void EmailSignUpFocusGained(java.awt.event.FocusEvent evt) {
        if (EmailSignUp.getText().equals(anObject: "Email address...")) {
            EmailSignUp.setText(t: "");
            EmailSignUp.setForeground(new Color(r: 51, g: 51, b: 51));
        }
    }

    private void EmailSignUpFocusLost(java.awt.event.FocusEvent evt) {
        if (EmailSignUp.getText().equals(anObject: "")) {
            EmailSignUp.setText(t: "Email address...");
            EmailSignUp.setForeground(new Color(r: 51, g: 51, b: 51));
        }       // TODO add your handling code here:
    }
```

Sign Up Button:

In this field check all the field is work or not.If fill then insert on the table then set null after otherwise prompth fill

```
    private void SignUpButtonActionPerformed(java.awt.event.ActionEvent evt) {
        if (FirstNameSignUp.getText().equals(anObject: "")){
            JOptionPane.showMessageDialog(parentComponent: this, message: "Insert first name");
            return;
        }

        if (LastNameSignUp.getText().equals(anObject: "")){
            JOptionPane.showMessageDialog(parentComponent: this, message: "Insert last name");
            return;
        }

        if (EmailSignUp.getText().equals(anObject: "")){
            JOptionPane.showMessageDialog(parentComponent: this, message: "Enter email");
            return;
        }

        if (PasswordSignUp.getText().equals(anObject: "")){
            JOptionPane.showMessageDialog(parentComponent: this, message: "Enter password");
            return;
        }

        if (PasswordSignUp.getText().length()<8){
            JOptionPane.showMessageDialog(parentComponent: this, message: "Password must be at least 8 digit");
            return;
        }
```

```
    if (!MaleSignUp.isSelected() && !FemaleSignUp.isSelected() && !OtherSignUp.isSelected()){
        JOptionPane.showMessageDialog(parentComponent: this, message: "Select gender");
        return;
    }

    if (!AgreeTerms.isSelected()){
        JOptionPane.showMessageDialog(parentComponent: this, message: "You must agree to the terms and condi
        return;
    }

    for(Profile p:Profile.Profiles){
        if(p.getUsername().equals(anObject: EmailSignUp.getText())){
            JOptionPane.showMessageDialog(parentComponent: this, message: "This email is already in use");
            return;
        }
    }
    String username, email, password, gender;

    username = FirstNameSignUp.getText() + " " + LastNameSignUp.getText();
    email = EmailSignUp.getText();
    password = PasswordSignUp.getText();
    gender = "?";
    if(MaleSignUp.isSelected()){
        gender = "Male";
    }else if(FemaleSignUp.isSelected()){
```

If all the input boxes is filled then go to the client panel.

```
    }else if(FemaleSignUp.isSelected()){
        gender = "Female";
    }else if(OtherSignUp.isSelected()){
        gender = "Other";
    }

    Client p = new Client(username, email, password, gender, balance: 0, loan:0, requestedLoan:0);
    LoggedInProfile = p;
    clearSignUpPanel();
    updateClientValues();
    SignUpPanel.setVisible(aFlag: false);
    ClientPanel.setVisible(aFlag: true);
}

public void clearSignUpPanel(){
    FirstNameSignUp.setText(t: "First name...");
    LastNameSignUp.setText(t: "Last name...");
    EmailSignUp.setText(t: "Email address...");
    PasswordSignUp.setText(t: "New password...");
    MaleSignUp.setSelected(b: false);
    FemaleSignUp.setSelected(b: false);
    OtherSignUp.setSelected(b: false);
    AgreeTerms.setSelected(b: false);
}
```

I Have an Account Button:

```
    private void GoToLoginActionPerformed(java.awt.event.ActionEvent evt) {
        clearSignUpPanel();
        LoginPanel.setVisible(aFlag: true);
        SignUpPanel.setVisible(aFlag: false);
    }
```

Profile Class:

In this class create profile for the client,admin.The method for deposit,withdraw,loan accept,decline,repay loan,reject loan etc.

```java
package bankmanagement;
import java.util.ArrayList;

public abstract class Profile {
    static ArrayList<Profile> Profiles = new ArrayList<>();

    private String username, email, password, gender;

    public Profile(String username, String email, String password, String gender) {
        this.username = username;
        this.email = email;
        this.password = password;
        this.gender = gender;
        Profiles.add(e: this);
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String Gender) {
        this.gender = Gender;
    }
}

class Client extends Profile {
    private int balance, loan, requestedLoan;
    private final ArrayList<Transaction> Transactions;
    public Client(String username, String email, String password, String gender, int balance, int loan, int requestedLoan) {
        super(username, email, password, gender);
        this.balance = balance;
        this.loan = loan;
        this.requestedLoan = requestedLoan;
        Transactions = new ArrayList<>();
    }

    public ArrayList<Transaction> getTransactions() {
```

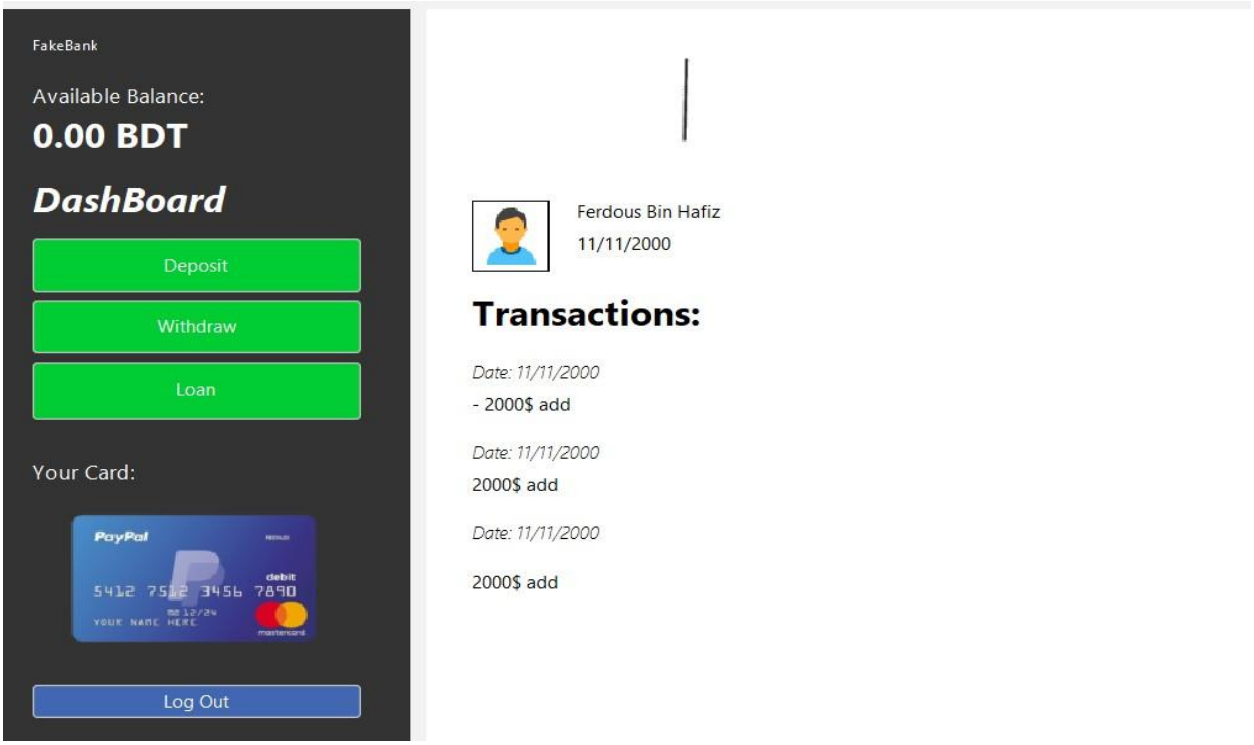
```
    public int getRequestedLoan() {
        return requestedLoan;
    }

    public void setRequestedLoan(int requestedLoan) {
        this.requestedLoan = requestedLoan;
    }

    public int getBalance() {
        return balance;
    }

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

    public int getLoan() {
        return loan;
    }

    public void setLoan(int loan) {
        this.loan = loan;
    }

    public void deposit(int Amount){
        balance += Amount;
        Transactions.add(index:0, new Transaction(Amount, Type: "Deposit"));
    }
```

```
        Transactions.add(index:0, new Transaction(Amount, Type: "Deposit"));
    }

    public void withdraw(int Amount){
        balance -= Amount;
        Transactions.add(index:0, new Transaction(Amount, Type: "Withdraw"));
    }

    public void borrow(int Amount){
        balance += Amount;
        loan += Amount;
        Transactions.add(index:0, new Transaction(Amount, Type: "Loan"));
    }

    public void repayLoan(int Amount){
        balance -= Amount;
        loan -= Amount;
        Transactions.add(index:0, new Transaction(Amount, Type: "Repayment"));
    }

    public void addTransaction(Transaction T){
        Transactions.add(e:T);
    }
}

class Admin extends Profile {
    static ArrayList<Client> LoanRequests = new ArrayList<>();
    public Admin(String username, String email, String password, String gender) {
        super(username, email, password, gender);
    }
```

```java
class Admin extends Profile {
    static ArrayList<Client> LoanRequests = new ArrayList<>();
    public Admin(String username, String email, String password, String gender) {
        super(username, email, password, gender);
    }

    public ArrayList<Client> getLoanRequests(){
        return LoanRequests;
    }

    public void acceptLoan(Client p){
        p.setBalance(p.getBalance()+p.getRequestedLoan());
        p.setLoan(p.getLoan()+p.getRequestedLoan());
        p.addTransaction(new Transaction(Amount:p.getRequestedLoan(), Type: "Loan"));
        p.setRequestedLoan(requestedLoan: 0);
        LoanRequests.remove(o: p);
    }

    public void rejectLoan(Client p){
        p.setRequestedLoan(requestedLoan: 0);
        LoanRequests.remove(o: p);
    }

    public void removeProfile(Client p){
        if(LoanRequests.contains(o: p)){
            LoanRequests.remove(o: p);
        }
        Profile.Profiles.remove(o: p);
    }
```

Account JFrame:

In this JFrame Deposit,withdraw,loan and log out use JButton other design use jLebal



Deposit Button:

In this button under deposit money add function.

```
    private void DepositProceedActionPerformed(java.awt.event.ActionEvent evt) {
        if(DepositAmount.getText().equals(anObject:"")){
            JOptionPane.showMessageDialog(parentComponent:this, message:"Enter Amount");
            return;
        }
        int Amount;
        try{
            Amount = Integer.parseInt(s:DepositAmount.getText());
        }catch(NumberFormatException e){
            JOptionPane.showMessageDialog(parentComponent:this, message:"Invalid Amount");
            return;
        }
        if(Amount<=0){
            JOptionPane.showMessageDialog(parentComponent:this, message:"Invalid Amount");
            return;
        }
        ((Client)LoggedInProfile).deposit(Amount);
        updateClientValues();
        JOptionPane.showMessageDialog(parentComponent:this, message:"Deposit Complete");
    }
```

Withdraw Button:

In this method for witdraw loan.

```
    private void WithdrawProceedActionPerformed(java.awt.event.ActionEvent evt) {
        if(WithdrawAmount.getText().equals(anObject:"")){
            JOptionPane.showMessageDialog(parentComponent:this, message:"Enter Amount");
            return;
        }
        int Amount;
        try{
            Amount = Integer.parseInt(s:WithdrawAmount.getText());
        }catch(NumberFormatException e){
            JOptionPane.showMessageDialog(parentComponent:this, message:"Invalid Amount");
            return;
        }
        if(Amount<=0){JOptionPane.showMessageDialog(parentComponent:this, message:"Invalid Amount"); r
        if(Amount>((Client)LoggedInProfile).getBalance()){JOptionPane.showMessageDialog(parentCompone
        ((Client)LoggedInProfile).withdraw(Amount);
        updateClientValues();
        JOptionPane.showMessageDialog(parentComponent:this, message:"Deposit Complete");
    }

    private void LoanProceedActionPerformed(java.awt.event.ActionEvent evt) {
        if(LoanAmount.getText().equals(anObject:"")){
```

Loan Button:

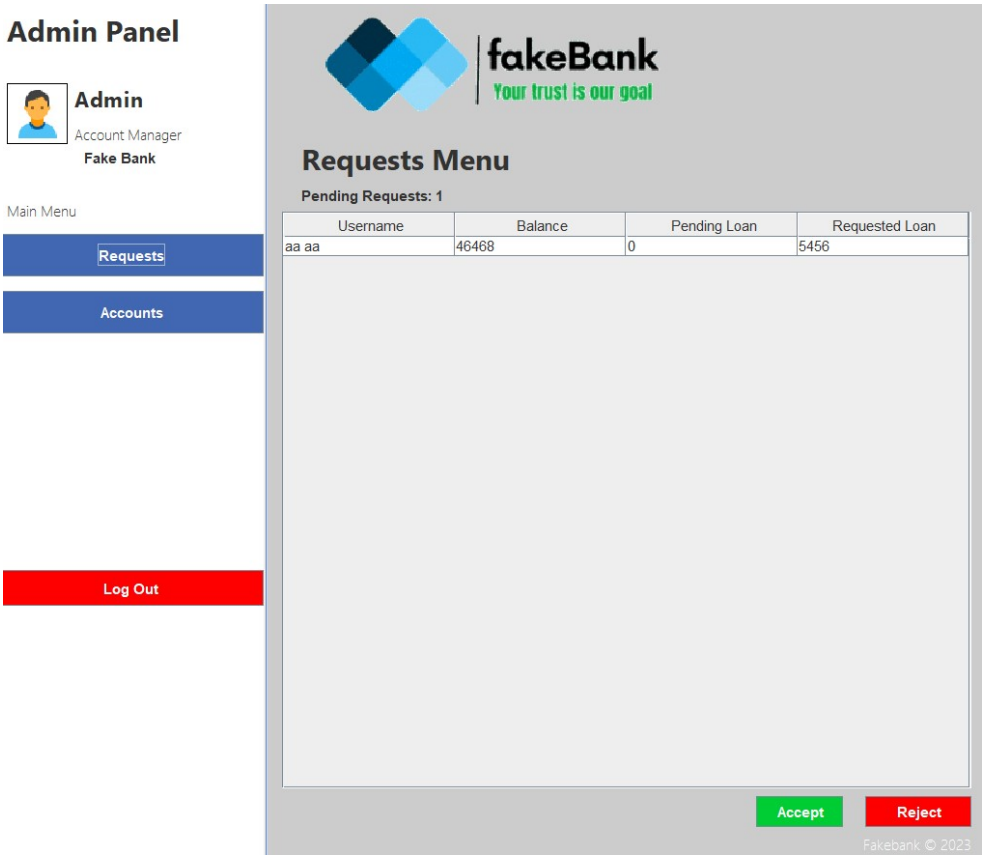
```
private void LoanProceedActionPerformed(java.awt.event.ActionEvent evt) {
    if(LoanAmount.getText().equals(anObject:"")){
        JOptionPane.showMessageDialog(parentComponent: this, message: "Enter Amount");
        return;
    }
    int Amount;
    try{
        Amount = Integer.parseInt(s: LoanAmount.getText());
    }catch(NumberFormatException e){
        JOptionPane.showMessageDialog(parentComponent: this, message: "Invalid Amount");
        return;
    }
    if(Amount>0){
        if(Amount + ((Client)LoggedInProfile).getLoan()>5000){
        int result = JOptionPane.showConfirmDialog(parentComponent: this, message: "[Maximum Loan is 5000]\nWould you like to request for a larger loan?", title: "Loan Re
        if(result == 0){
            ((Client)LoggedInProfile).setRequestedLoan(requestedLoan:Amount);
            Admin.LoanRequests.add((Client)LoggedInProfile);
            JOptionPane.showMessageDialog(parentComponent: this, message: "Loan Requested");
        }
        return;
    }
        ((Client)LoggedInProfile).borrow(Amount);
        updateClientValues();
        JOptionPane.showMessageDialog(parentComponent: this, message: "Loan Accepted");
    }else if(Amount<0){
        if(((Client)LoggedInProfile).getLoan()< Amount*-1){
            JOptionPane.showMessageDialog(parentComponent: this, message: "Amount greater than pending loan");
            return;
        }
        ((Client)LoggedInProfile).repayLoan(Amount*-1);
        JOptionPane.showMessageDialog(parentComponent: this, message: "Loan paid");
    }else{
        JOptionPane.showMessageDialog(parentComponent: this, message: "Invalid Amount");
    }
```

Transaction class:

```
package bankmanagement;

public class Transaction {
    private int Amount;
    private String Type;

    public Transaction(int Amount, String Type) {
        this.Amount = Amount;
        this.Type = Type;
    }

    public int getAmount() {
        return Amount;
    }

    public void setAmount(int Amount) {
        this.Amount = Amount;
    }

    public String getType() {
        return Type;
    }

    public void setType(String Type) {
        this.Type = Type;
    }
}
```

Admin Panel:

In this panel 2 button one for loan request and another for accont also accept and request.



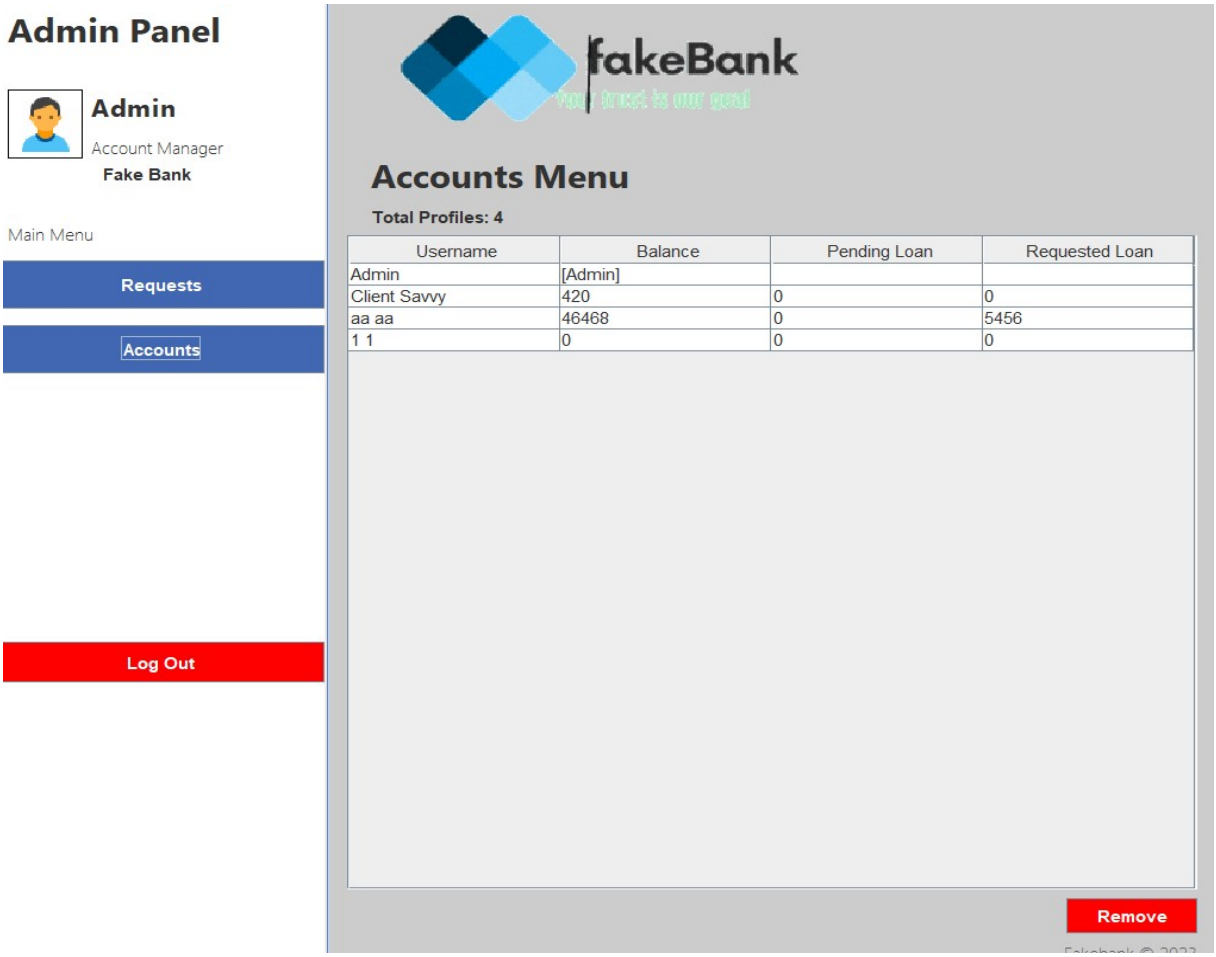
Accept Loan:

```
private void AcceptRequestActionPerformed(java.awt.event.ActionEvent evt) {
    int r[] = AllRequestTable.getSelectedRows();
    Admin admin = (Admin)LoggedInProfile;
    ArrayList<Client> list = admin.getLoanRequests();
    for(int i: r){
        admin.acceptLoan(p:list.get(index: i));
    }
    refreshRequests();
}

Profile LoggedInProfile = null;
public void initializeUIStartUp(){
    LoginPanel.setVisible(aFlag: true);
    SignUpPanel.setVisible(aFlag: false);
    ClientPanel.setVisible(aFlag: false);
    AdminPanel.setVisible(aFlag: false);

}
```

Reject Loan:

```
private void RejectRequestActionPerformed(java.awt.event.ActionEvent evt) {
    int r[] = AllRequestTable.getSelectedRows();
    Admin admin = (Admin)LoggedInProfile;
    ArrayList<Client> list = admin.getLoanRequests();
    for(int i: r){
        admin.rejectLoan(p:list.get(index: i));
    }
    refreshRequests();
}

private void RemoveProfileActionPerformed(java.awt.event.ActionEvent evt) {
    int r[] = AllProfileTable.getSelectedRows();
    if(r.length>1){
        JOptionPane.showMessageDialog(parentComponent:this, message:"Multiple selection detected");
        return;
    }else if(r.length == 0){
        JOptionPane.showMessageDialog(parentComponent:this, message:"No selection detected");
        return;
```

Account Button:



Remove Account:

```
private void RemoveProfileActionPerformed(java.awt.event.ActionEvent evt) {
    int r[] = AllProfileTable.getSelectedRows();
    if(r.length>1){
        JOptionPane.showMessageDialog(parentComponent: this, message: "Multiple selection detected");
        return;
    }else if(r.length == 0){
        JOptionPane.showMessageDialog(parentComponent: this, message: "No selection detected");
        return;
    }

    Admin admin = (Admin)LoggedInProfile;
    Profile p = Profile.Profiles.get(r[0]);
    if(p instanceof Admin){
       JOptionPane.showMessageDialog(parentComponent: this, message: "Selected profile is an admin profile
       return;
    }
    int result = JOptionPane.showConfirmDialog(parentComponent: this, message: "Are you sure?", title: "Cor
    if(result==0){
       admin.removeProfile((Client)p);
       refreshProfiles();
    }
}
```

Update Accont:

```
void updateClientValues(){
    Client p = (Client)LoggedInProfile;
    ClientName1.setText(text: p.getUsername());
    ClientName2.setText(text: p.getUsername());
    ClientName3.setText(text: p.getUsername());
    ClientName4.setText(text: p.getUsername());

    ClientBalance.setText(p.getBalance() + " BDT");

    ClientBalance1.setText("Current Balance: "+ p.getBalance() + " BDT");
    ClientBalance2.setText("Current Balance: "+ p.getBalance() + " BDT");
    ClientBalance3.setText("Current Balance: "+ p.getBalance() + " BDT");

    ClientLoan1.setText("Pending Loan: " + p.getLoan() + " BDT");
    ClientLoan2.setText("Pending Loan: " + p.getLoan() + " BDT");
    ClientLoan3.setText("Pending Loan: " + p.getLoan() + " BDT");

    TransactionDate1.setText(text: "");
    TransactionDate2.setText(text: "");
    TransactionDate3.setText(text: "");

    Transaction1.setText(text: "");
    Transaction2.setText(text: "");
    Transaction3.setText(text: "");
    javax.swing.JLabel T[] = {Transaction1, Transaction2, Transaction3};
    javax.swing.JLabel D[] = {TransactionDate1, TransactionDate2, TransactionDate3};

    ArrayList<Transaction> Transactions = p.getTransactions();
    int i;
    for(i=0; i<Math.min(a: 3, b: Transactions.size()); i++){
```

```
            LastDeposit.setText(v.getAmount()+" BDT");
            LastDepositDate.setText(text: "11/09/2001");
            break;
        }
    }

    LastWithdraw.setText(text: "");
    LastWithdrawDate.setText(text: "");
    for(Transaction v : Transactions){
        if("Withdraw".equals(anObject: v.getType())){
            LastWithdraw.setText(v.getAmount()+" BDT");
            LastWithdrawDate.setText(text: "11/09/2001");
            break;
        }
    }

    LastLoan.setText(text: "");
    LastLoanDate.setText(text: "");
    for(Transaction v : Transactions){
        if("Loan".equals(anObject: v.getType())){
            LastLoan.setText(v.getAmount()+" BDT");
            LastLoanDate.setText(text: "11/09/2001");
            break;
        }
    }
}
public void updateAdminValues(){
    AdminName.setText(text: LoggedInProfile.getUsername());
}
void clearClientFrames(){
```

Refresh Profile Request:

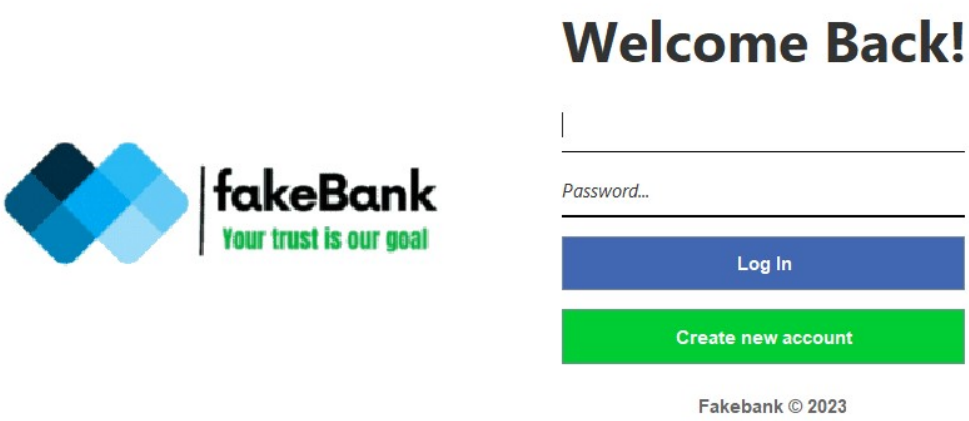
```
public void refreshProfiles(){
    DefaultTableModel dtm = (DefaultTableModel)AllProfileTable.getModel();
    dtm.setRowCount(rowCount: 0);
    Object r[] = new Object[4];
    for(Profile p: Profile.Profiles){
        r[0] = p.getUsername();
        if (p instanceof Client c){
            r[1] = c.getBalance();
            r[2] = c.getLoan();
            r[3] = c.getRequestedLoan();
        }else if(p instanceof Admin){
            r[1] = "[Admin]";
            r[2] = "";
            r[3] = "";
        }
        dtm.addRow(rowData:r);
    }
    TotalProfiles.setText("Total Profiles: "+ Profile.Profiles.size());
}
```

Refresh Loan Request:

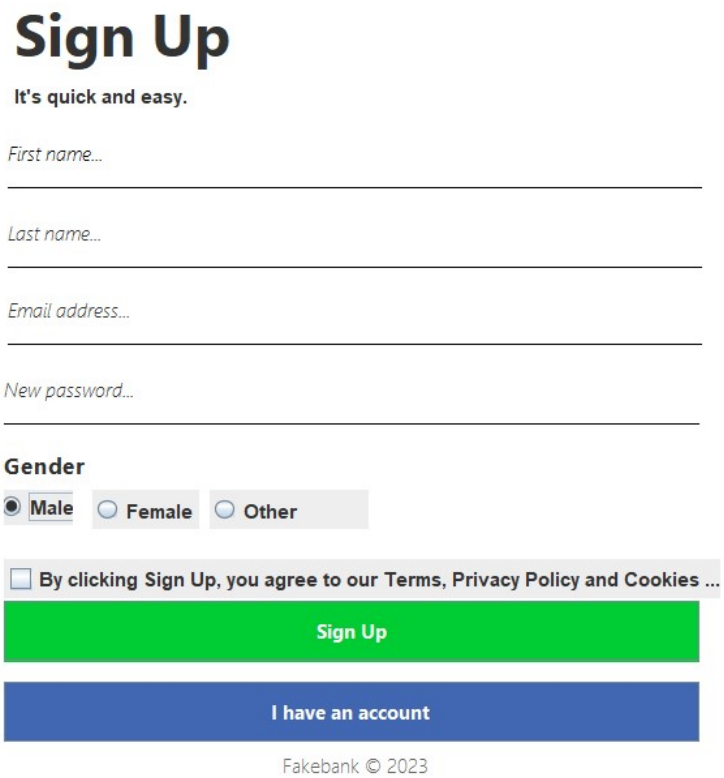
```
public void refreshRequests(){
    DefaultTableModel dtm = (DefaultTableModel)AllRequestTable.getModel();
    dtm.setRowCount(rowCount: 0);
    Object r[] = new Object[4];
    Admin a = (Admin)LoggedInProfile;

    for(Profile p: a.getLoanRequests()){
        r[0] = p.getUsername();
        if (p instanceof Client c){
            r[1] = c.getBalance();
            r[2] = c.getLoan();
            r[3] = c.getRequestedLoan();
        }
        dtm.addRow(rowData:r);
    }
    PendingRequests.setText("Pending Requests: " + ((Admin)LoggedInProfile).getLoanRequests().size());
}
```

Output:

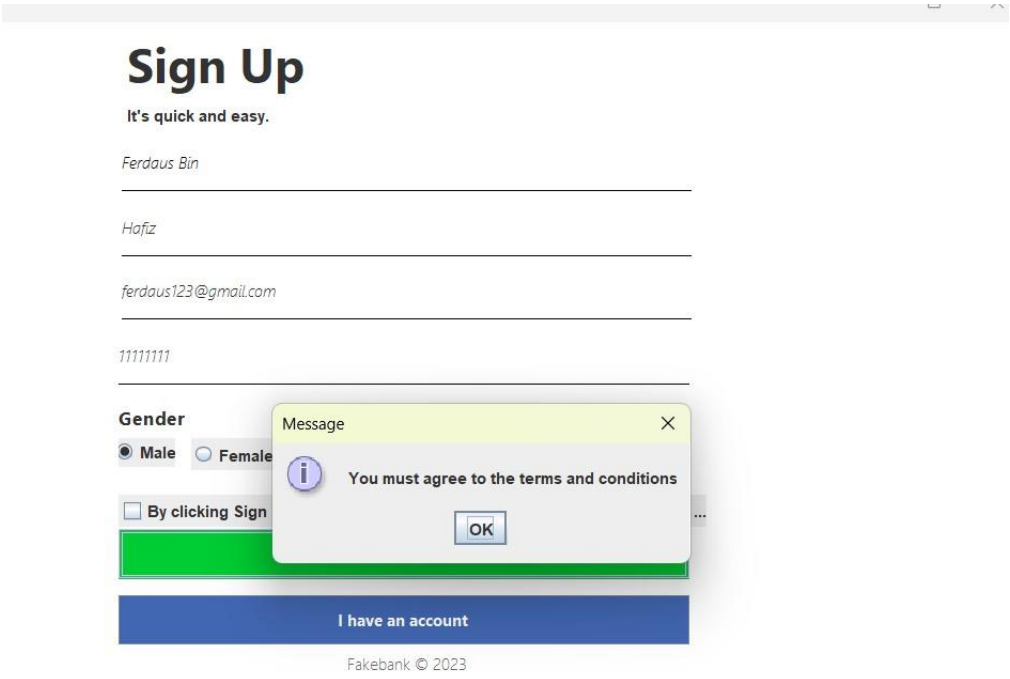
When run the project first interface



Then firstly I want to create new account so that click on create new account then com this form

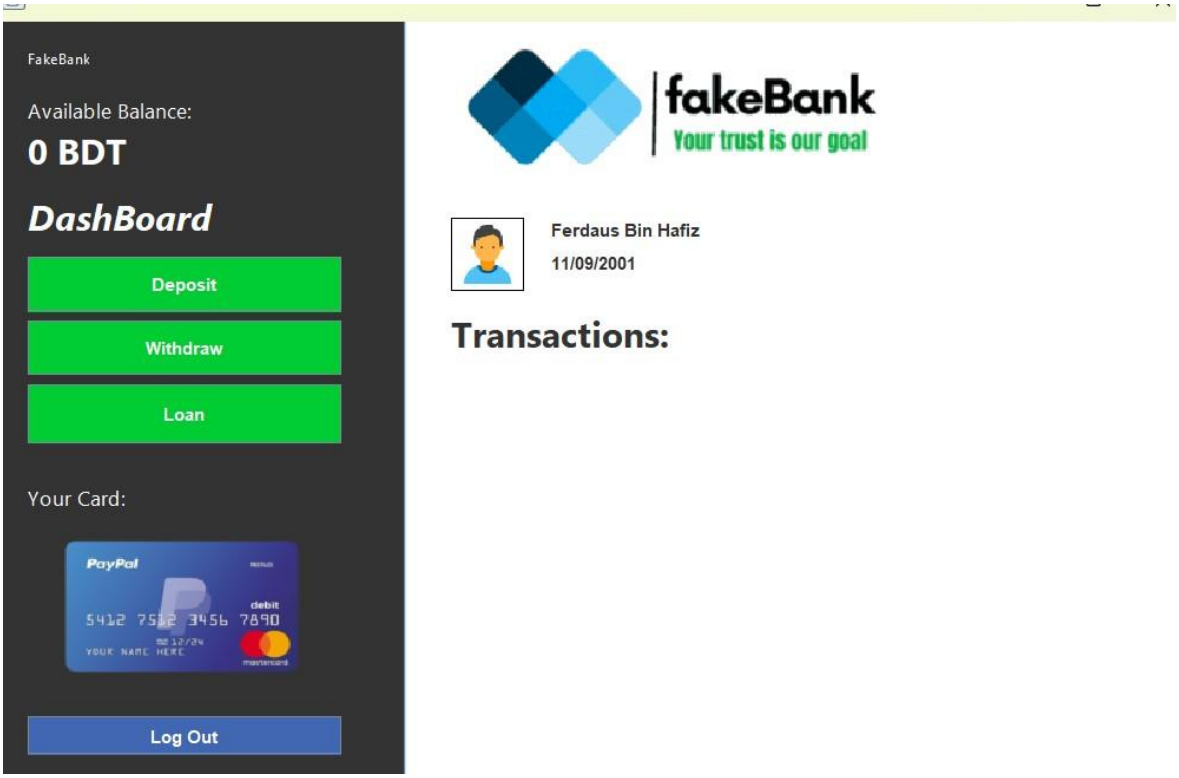


If I do not fill any gap message show fill all the gap

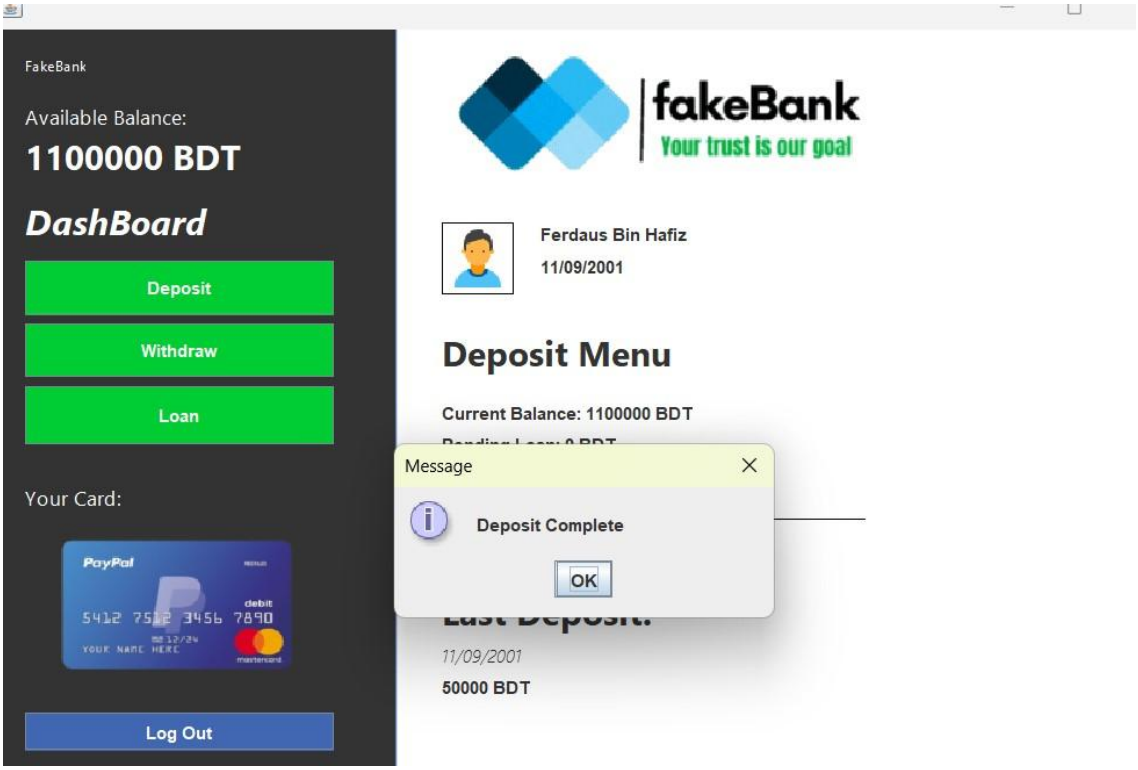


If the password is less than 8 digit show must be 8 digit

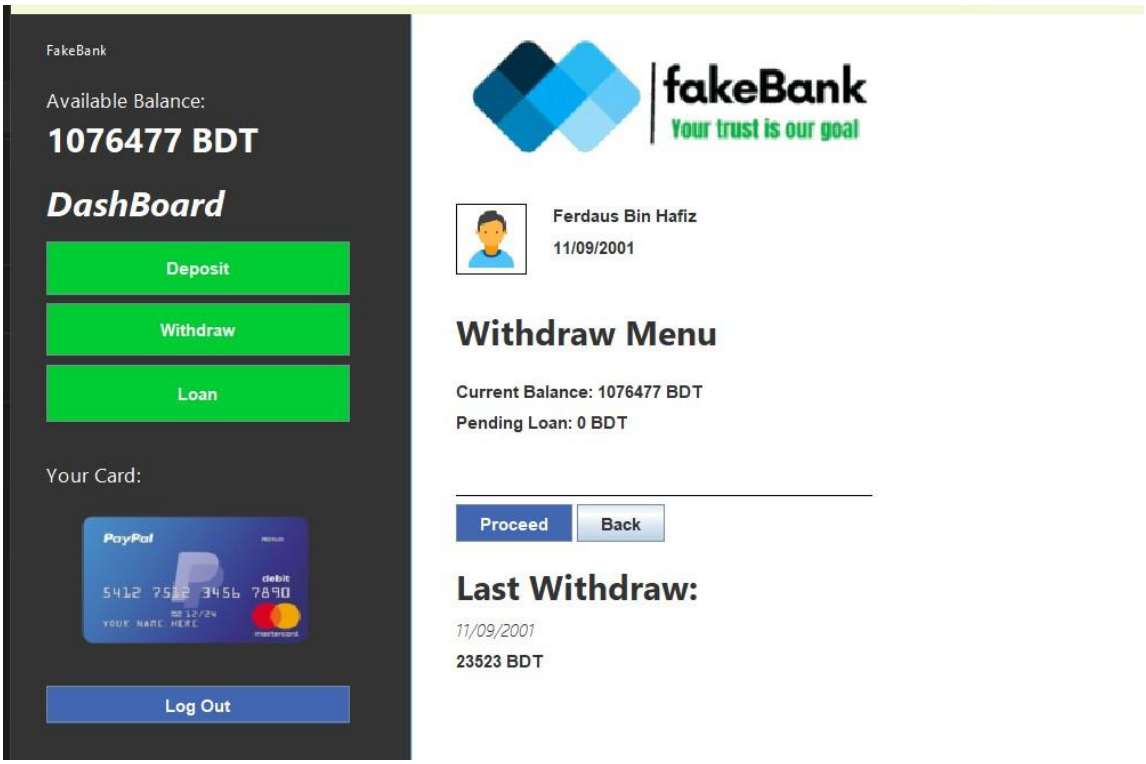
If fill all the gap and click sign up it goes to the account information dashboard



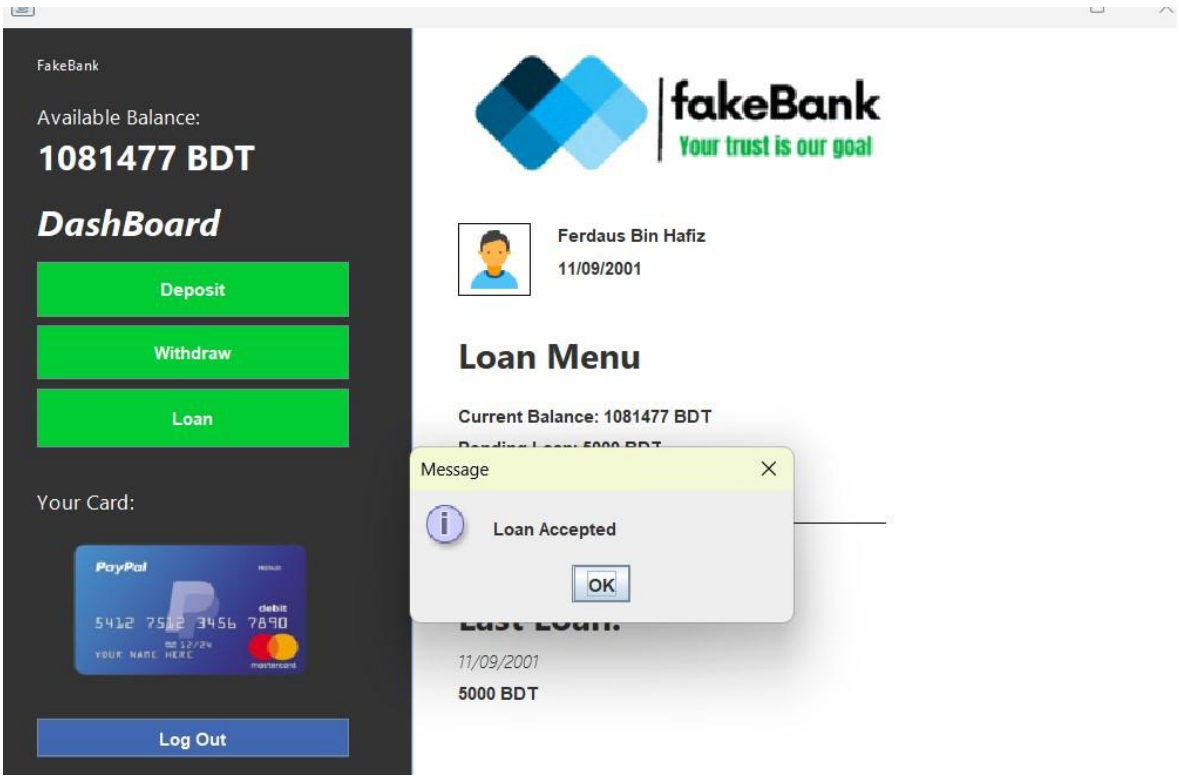
If I choose the deposit panel it can skope to set amount to deposit and add in the balance



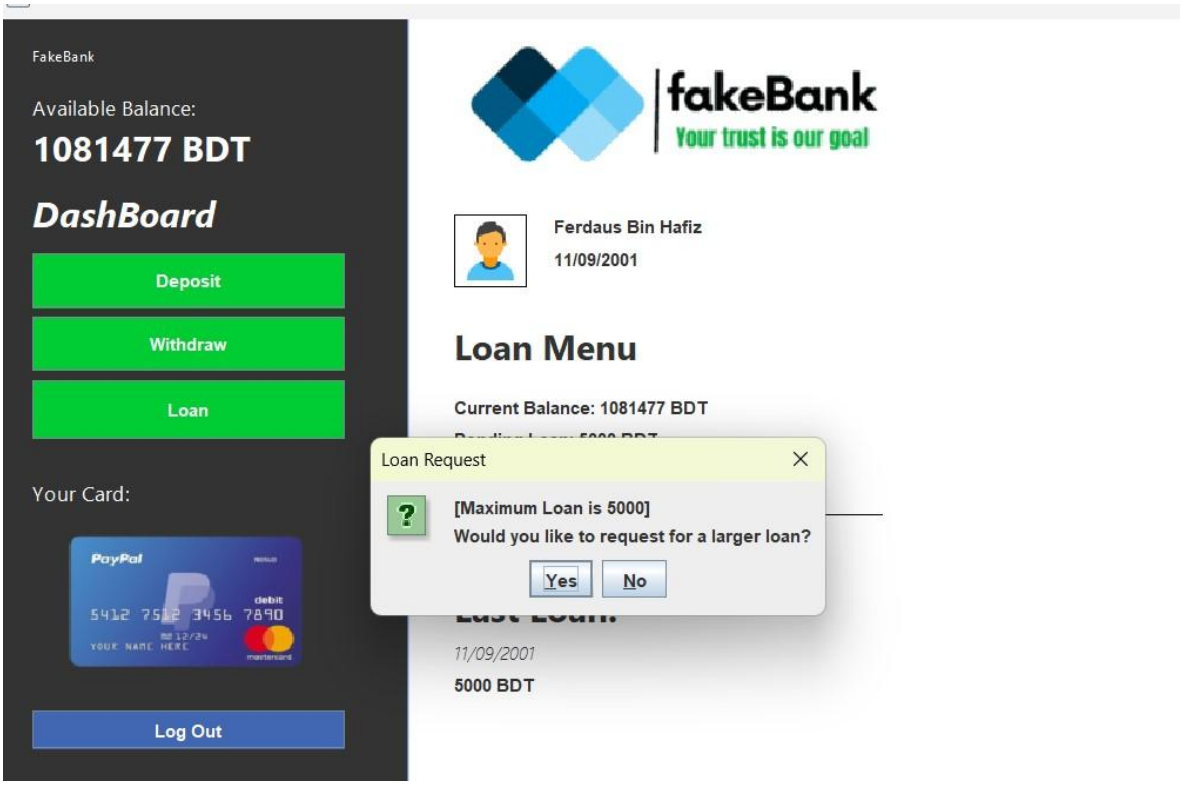
If I choose the Withfraw option then it clear the deposit money input from the deposit input textfield



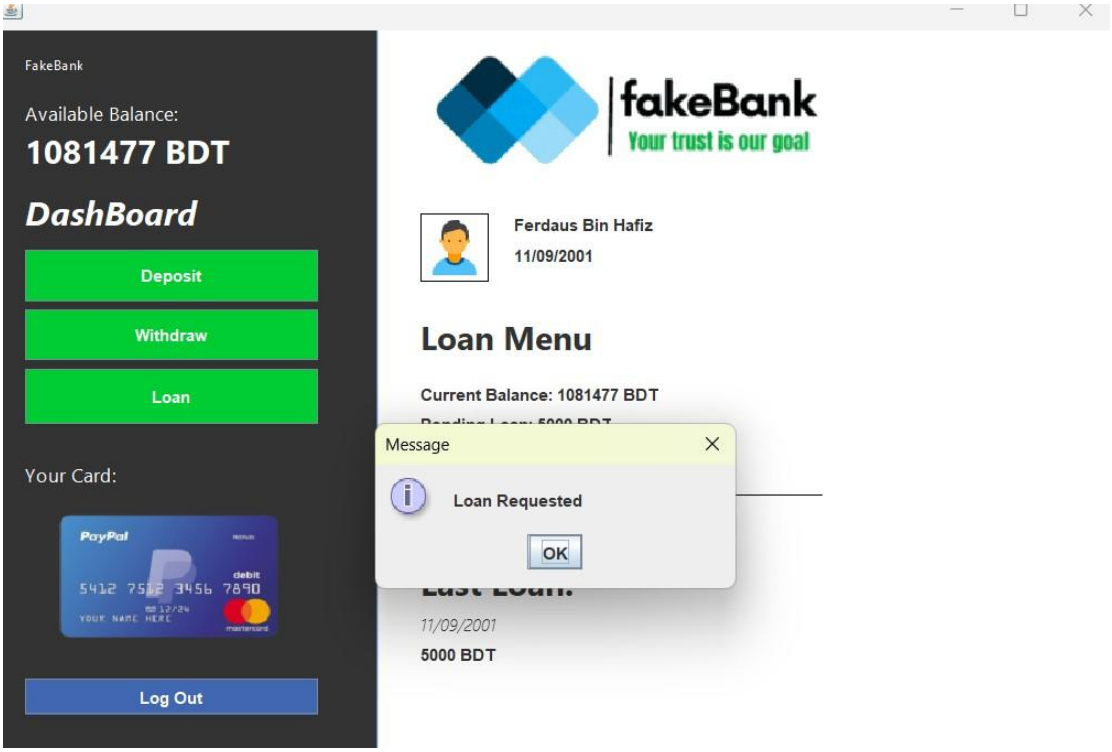
If I choose the loan then give the chance to get loan amount.



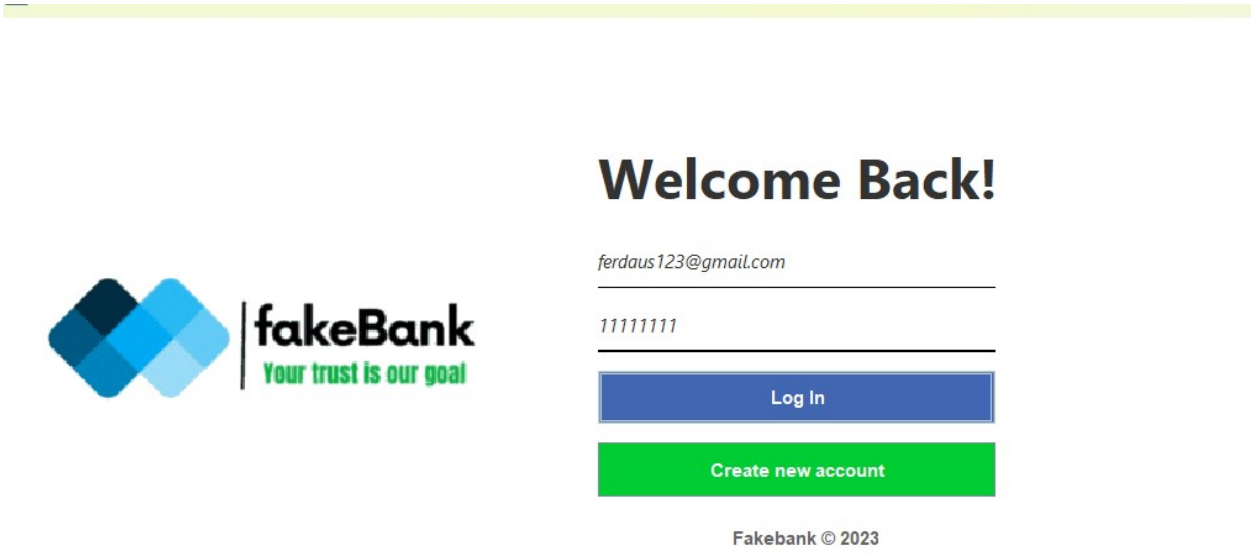
If client try to get more than 5000 then he can not directly get loan then he can request for loan to the admin.



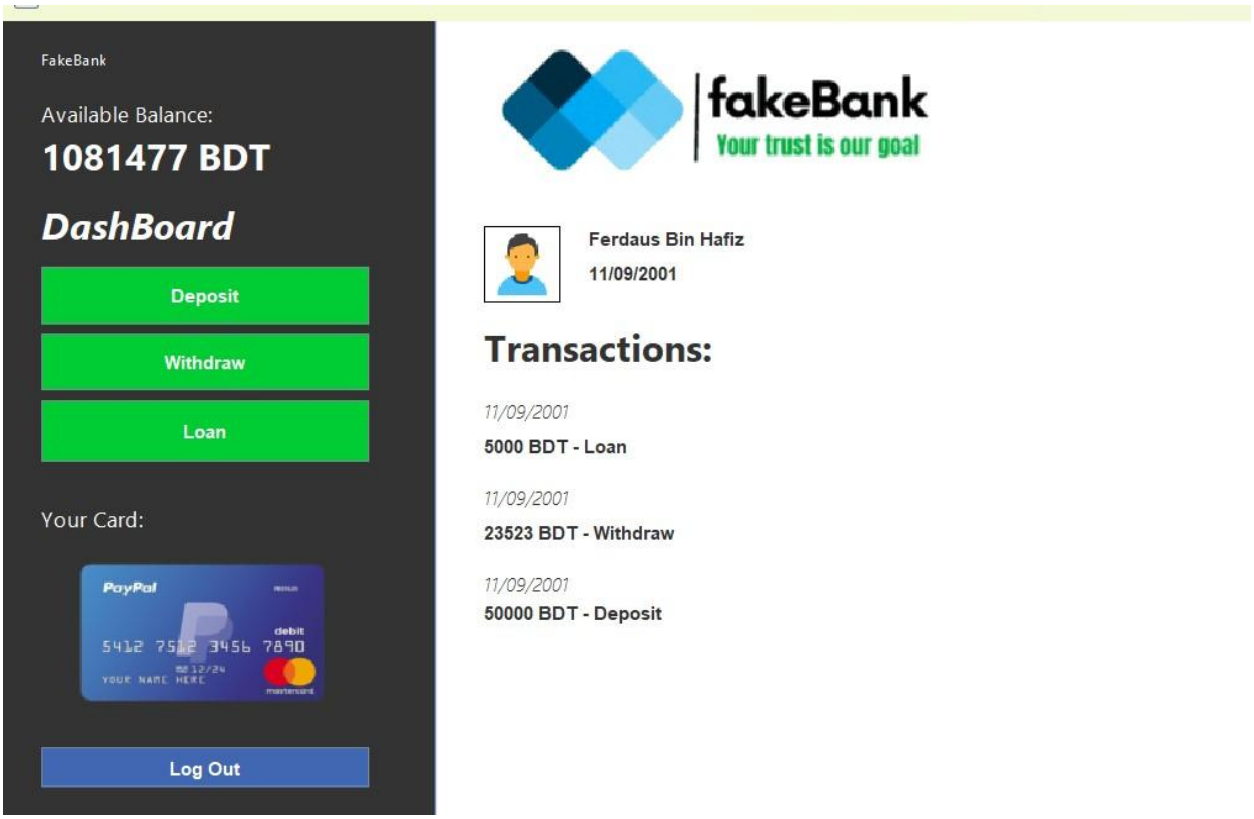
If client select yes then the request goes to the admin for approval



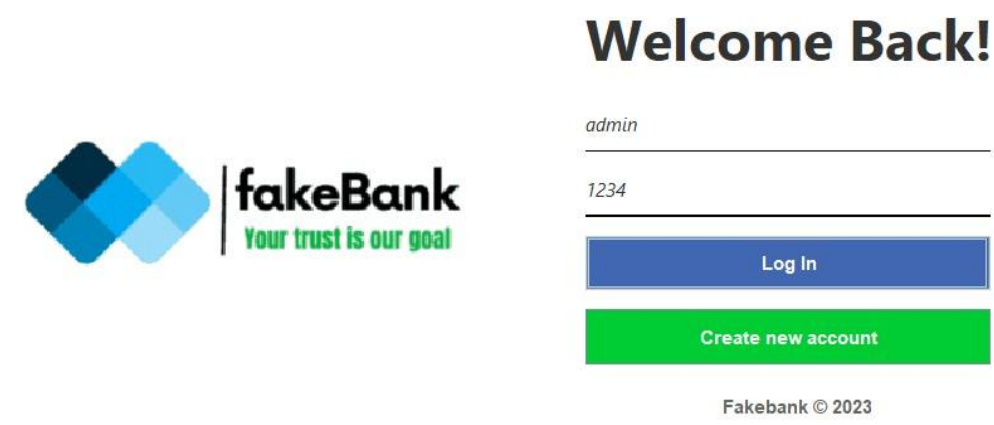
If client choose the log out it goes to the login in panel and try with the create account email and password



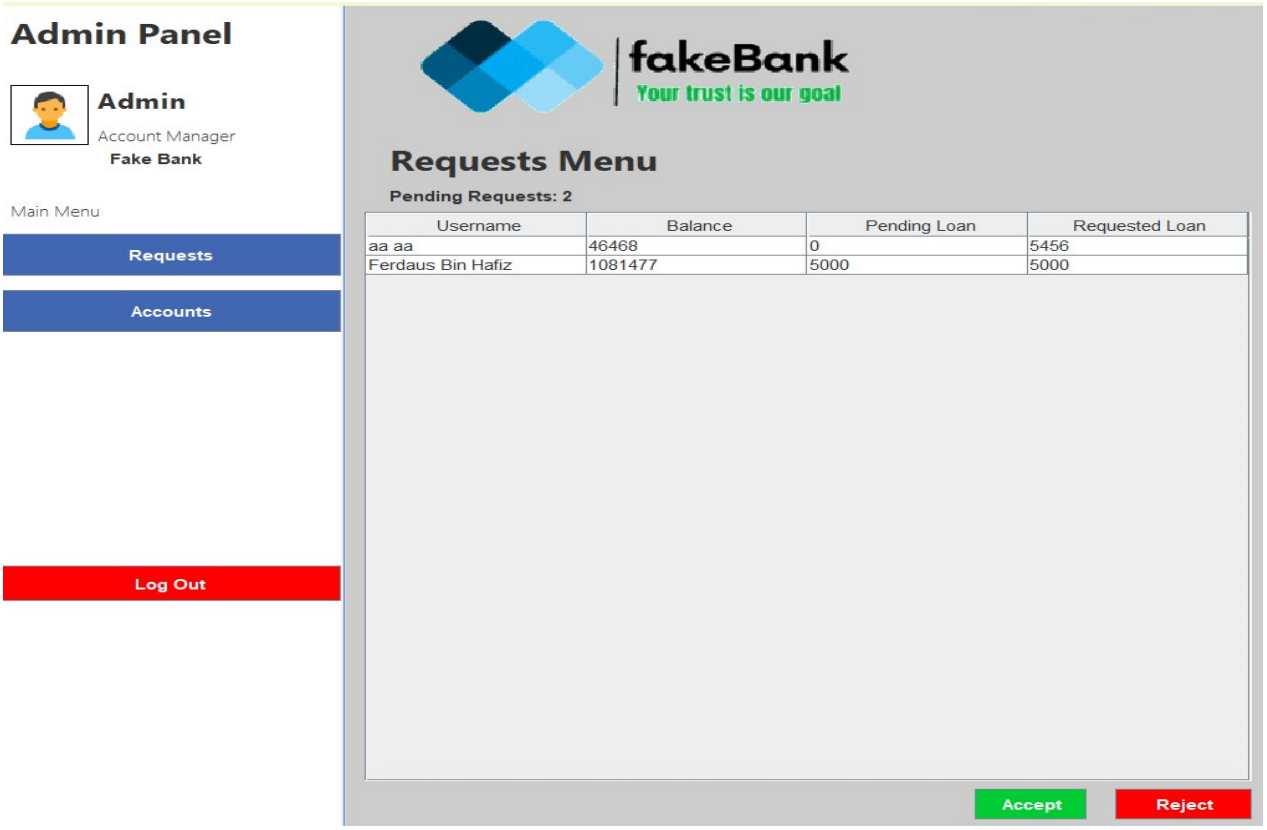
Then click the login panel to login.After login it goes the account dashboard



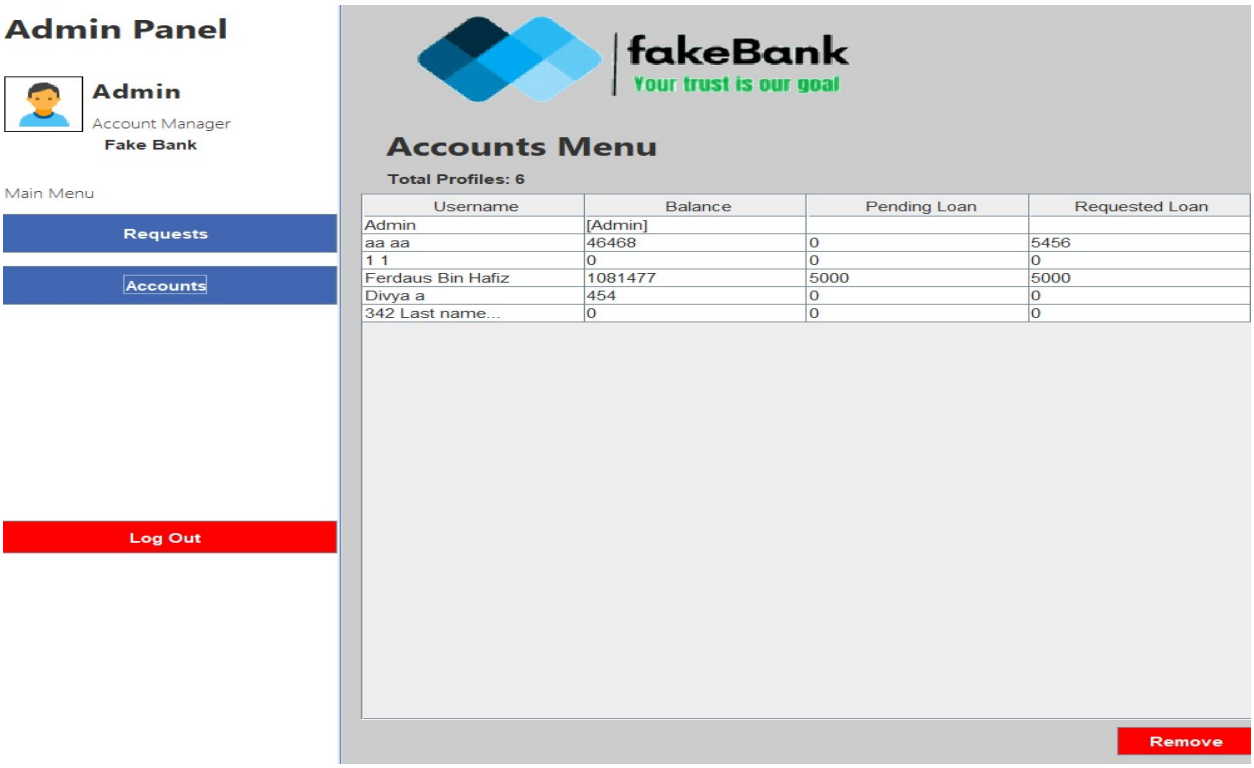
Then logout again and try to admin login



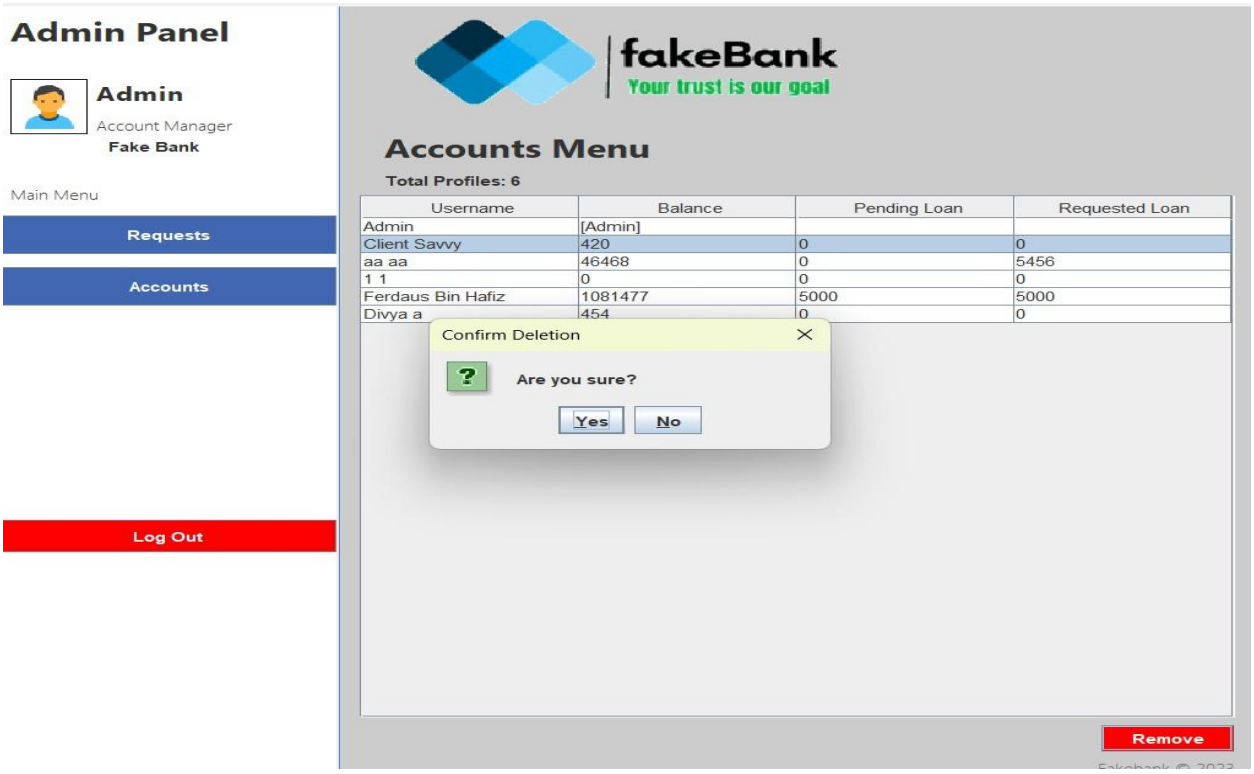
After login it goes to the account list.There two option loan request and account.In this part show whose request for loan.If accept then the amount add in the balance else reject than cannot add in the account.



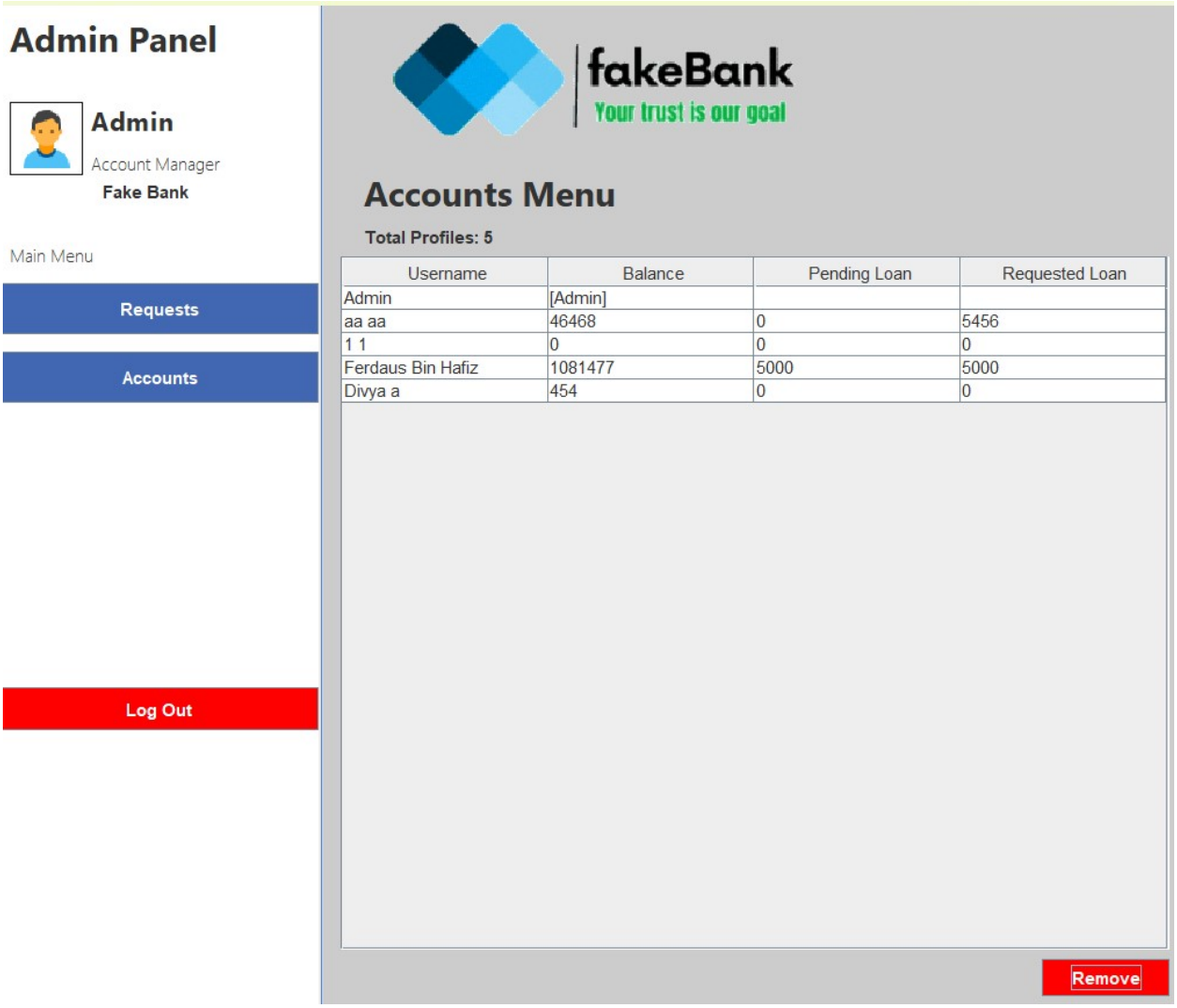
If choose account then show all account.



Then select any row and then click remove then the account will be delete but before message me are you sure.if yes then delete otherwisw not



If yes then delete from list



COMPLETE INSIGHT

The following project holds only but a portion of the whole idea I pictured. I wanted to add more methods for profile class, such as checking all the posts made by a specific profile, all the liked posts by a profile, all the posts shared on a profile, option to delete a profile. On other-hand for posts I wanted to implement similar methods displaying all the profiles that liked a post, all the profiles that shared a post, option to delete a post. Some methods could have been improved, for example in the log into profile function using password to log in, if input password does not

match the profile password, prevent user from logging in. Other plans to improve the project further could have been made, but due to short amount of time the project had to be left simple and less complex.

CONCLUSION

In this report, we successfully implemented a Post superclass and TextPost, VideoPost subclass inheriting the post class. We made the Profile class to create posts using methods and user inputs. In social media class, we got 4 main methods that allows you to: create new account, log in to account, create post and browse posts. All the profiles and posts are organized and maintained using array list, for easing things up. Overall, the code successfully displayed all the features and properties of object-oriented programming in java, solving real world problem.