ABSTRACT

The ATM Machine project is a software application that emulates the functionalities of an Automated Teller Machine (ATM). The goal of this project is to develop a user-friendly and efficient system that enables users to perform common banking operations such as cash withdrawal, balance inquiry, fund transfer, and account management. The ATM Machine project utilizes Java programming language and incorporates key concepts such as object-oriented programming, data structures, and database management. The system implements a secure authentication mechanism to ensure that only authorized users can access their bank accounts.

This project aims to develop an ATM machine system using Java Swing, a modern GUI library for Java-based applications. The system is designed to provide essential functionalities such as account registration, login, balance inquiry, cash withdrawal, and balance update. The system is developed following software engineering principles to ensure that it is reliable, scalable, and secure.

The system is designed using object-oriented programming principles to make it easy to maintain and extend. The user interface is implemented using Java Swing, which provides a user-friendly and responsive experience for customers. Furthermore, the system is thoroughly tested using various test cases to ensure that it functions correctly and meets the requirements specified in the project.

In conclusion, this project showcases how to develop an ATM system using Java Swing, software engineering principles, and object-oriented programming. The project demonstrates the importance of using modern technologies and sound software engineering practices to develop robust and efficient systems.

Table of Contents

List of Figures

CHAPTER 1

INTRODUCTON

1.1 GENERAL

An Automated Teller Machine (ATM) is a self-service machine that enables bank customers to perform various financial transactions, including cash withdrawals, balance inquiries, fund transfers, and bill payments. The invention of the ATM machine has revolutionized banking by making it more convenient and accessible to people. In this article, we provide a general overview of the ATM machine, its components, and how it functions.

Overview of ATM Machine

The ATM machine is a computer-based system that provides self-service banking facilities to customers through a user-friendly graphical interface. The machine consists of several components, including a card reader, a keypad, a display screen, a cash dispenser, and a printer. These components work together to enable customers to perform various transactions.

When a customer approaches the machine, they must insert their ATM card into the card reader and enter their Personal Identification Number (PIN) using the keypad. The machine reads the customer's data from the magnetic stripe on the card and verifies the PIN with a secure database. Once the machine confirms the authenticity of the card and the PIN, it gives the user access to the account through a graphical interface.

The customer can choose from a range of options, including cash withdrawals, balance inquiries, fund transfers, and bill payments. The customer chooses the desired transaction by pressing the appropriate button on the keypad, and the machine processes the transaction. For cash withdrawals, the machine dispenses the requested cash from the cash dispenser, and for balance inquiries, the machine displays the account balance on the screen.

Components of ATM Machine:

1. Card Reader: The card reader is a device that reads the customer's ATM card and retrieves the data required for the transaction. The reader has a magnetic stripe reader for reading the data on the card.
2. Keypad: The keypad is a device that allows customers to input their PIN and other transaction-related information.
3. Display Screen: The display screen is a graphical interface that displays the transaction options and guides the customer through the process.
4. Cash Dispenser: The cash dispenser is a device that dispenses cash for cash withdrawals.
5. Printer: The printer is a device that prints transaction receipts for customers.

Functioning of ATM Machine

The ATM machine functions by communicating with a secure database through a network. The customer card is read by the magnetic stripe reader, and the PIN is verified by accessing the database. Once the user is authenticated, they are allowed to select from various transactions by pressing the corresponding options on the keypad. The machine fetches the requested data, displays it on the screen, and processes the transaction. The cash dispenser gives the requested cash, and the printer prints a receipt for the transaction.

Security is an essential consideration in ATM machines to prevent fraud and unauthorized access to customer accounts. The machine isolates the customer's transactions from other transactions and encrypts the data to keep it secure. The customer's account information is protected by a Personal Identification Number (PIN) to prevent unauthorized access to the account.

1.2 OBJECTIVE AND PROBLEM STATEMENT

1.2.1 OBJECTIVE

Our objective is to create a Java Swing interface for a software simulation of an ATM system that accurately mirrors the transactions of a physical ATM machine. Using coding best practices and harnessing the full potential of Java Swing features, we aim to develop a dependable and secure ATM system that provides easy and user-friendly access to banking services.

1.2.2 PROBLEM STATEMENT

To create a secure and user-friendly ATM system simulation using the Java Swing interface in line with physical ATM machines. We will utilize JAVA coding practices and Java Swing's features for accurate transaction processing.

CHAPTER 2

METHODOLOGY

2.1 ALGORITHMIC DETAILS

Algorithm:

1. Login Options:

- Ask User If He / She wants to create a new account using “Sign Up “option

- Or Login with there previous Account stored in System memory.

2. User Authentication:

- Prompt the user to enter their account number and PIN.

- Validate the input against a database of user accounts.

- If the input is correct, proceed to the next step. Otherwise, display an error message and ask the user to try again.

3. Menu Options:

- Display a menu of available options to the user, such as "Withdrawal", "Deposit", "Bank Details" "Change Pin”, etc.

- Prompt the user to select an option by entering a corresponding number.

4. Perform Actions based on Menu Selection:

- i) If the user selects **"Withdrawal"**:

- Prompt the user to enter the withdrawal amount.

- Check if the account balance is sufficient for the withdrawal.

- If yes, deduct the amount from the account balance. If no, display an error message.

- ii) If the user selects **"Deposit"**:

- Prompt the user to enter the deposit amount.

- Add the amount to the account balance.

- iii) If the user selects **"Bank Info":**

- Display the current account bank information to the user.

-iv) If the user selects **"Change Pin":**

- Changes the current Pin to the New Pin.

Flowchart: -

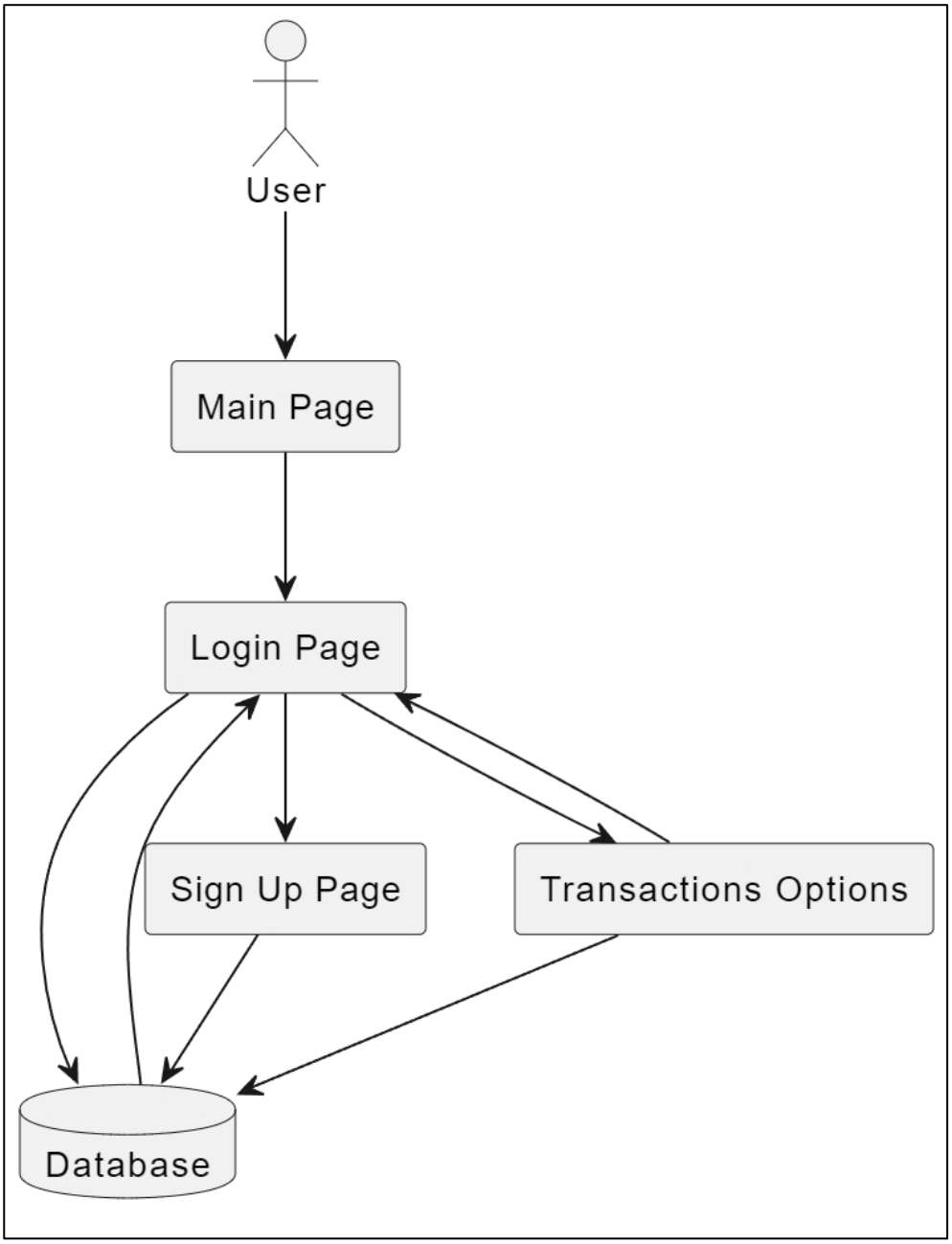


Fig 1: - Simple Working of ATM

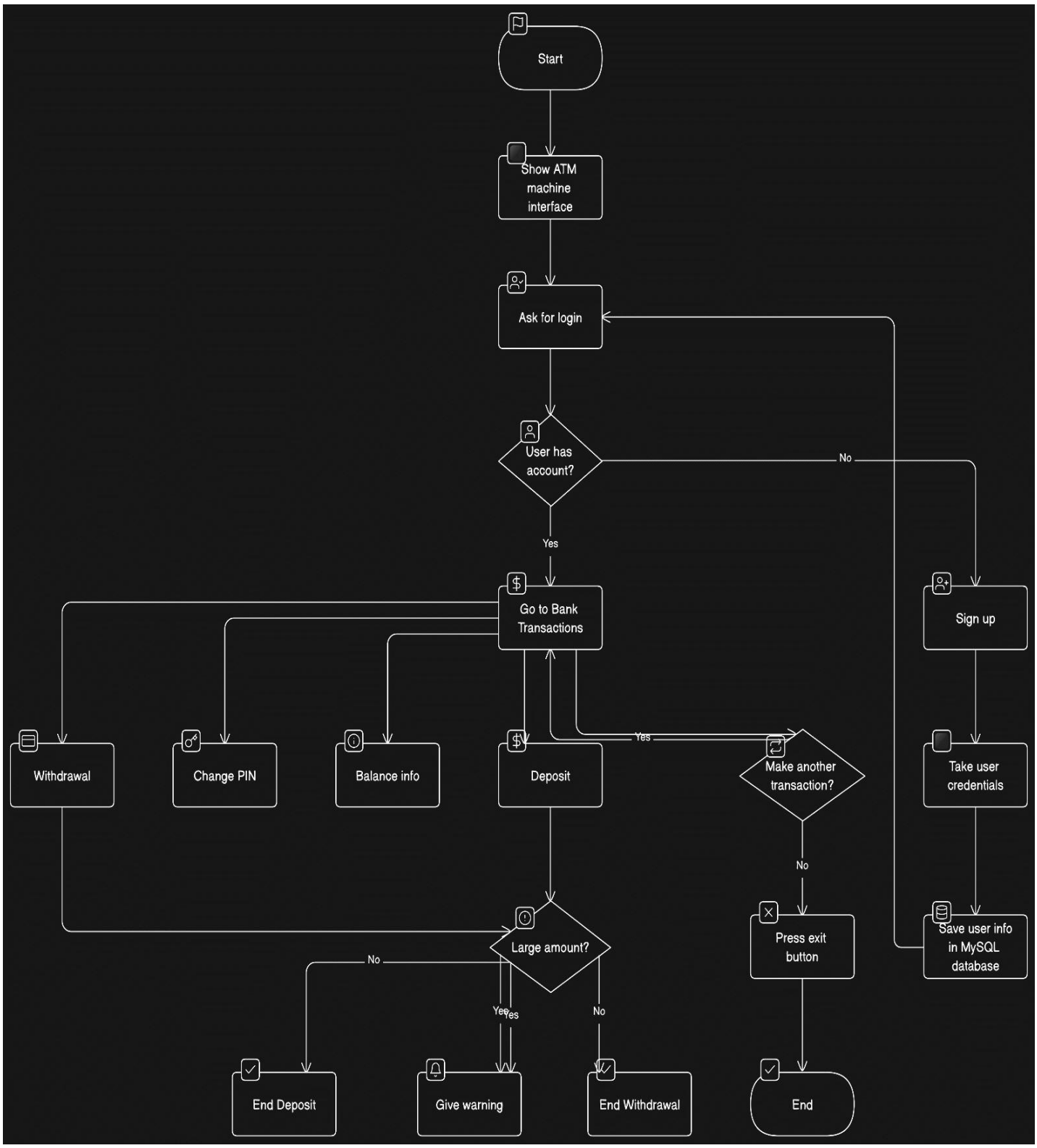


Fig 2: - Whole Working Process of App and its Connection

2.2 HARDWARE AND SOFTWARE REQUIREMENTS

2.2.1 HARDWARE REQUIREMENTS

1. RAM: 512 MB RAM
2. Hard Drive: 40 GB Hard Drive
3. Processor: Intel Core 2 Processor

2.2.2 SOFTWARE REQUIREMENTS

1. Java Runtime Environment (JRE): It is necessary to have the Java Runtime Environment installed on your system. Java SE Runtime Environment (JRE) is a free program that is required to run Java Swing applications.

2. Operating System: Java Swing is a cross-platform library, which means it can run on multiple operating systems, including Windows, Linux, and macOS.

3. Java Development Kit (JDK): If you are a developer and want to create or modify a Java Swing application, you need to install the Java Development Kit on your system.

2.3 DESIGN DETAILS:

1. JAVA-SWING: - We utilized Java Swing's basic components, such as JTextField, JFrame, JPanel, and JSpinner, to create an ATM machine project. Our window-based GUI application displayed account details, offering options for cash withdrawal or deposit, balance inquiry, and other features.

We used JTextFields for user input, received through their PIN or desired withdrawal/deposit amounts, and organized all components through a JFrame and JPanel, creating a consistent layout. JSpinner enabled users to choose transaction amounts efficiently while JLabel provided input descriptions.

Swing's layout managers, like BorderLayout, allowed for proper component spacing, making the interface user-friendly and intuitive. In conclusion, our project's use of basic Java Swing elements enabled us to create an efficient and functional ATM machine application in a short amount of time.

2. NETBEANS-IDE : - We used the NetBeans IDE's drag-and-drop feature to develop an ATM machine GUI with Java Swing. The pre-built components provided by the IDE were easily customized to fit the project's requirements. Event handling was implemented via NetBeans, which made their development simple and streamlined. JTextFields were utilized to capture user input and JSpinner components for deposit and withdrawal.

The JFrame container held all the components, while a JPanel was used for organizing them and providing consistency in layout. The JSpinner made for an intuitive, quick way for users to select their deposit/withdrawal amount, while the JLabel component was used to provide descriptions for the interface's different inputs and fields.

Swing's layout managers, particularly BorderLayout, helped regulate spacing between elements on the interface. We were able to create a capable and efficient ATM machine system utilizing Java Swing and NetBeans drag-and-drop features in a brief period of time.

3. MySQL: - We used SQL queries to create a database of user information, including their passwords and account balances. The login process involved a query to the database, checking the user's submitted password against the one stored in the database, allowing access if the credentials match. When the user logged in, their account balance was also retrieved from the database.

For balance updates, we used SQL commands in response to the user's deposit or withdrawal request. When the transaction was complete, the database was updated with the new account balance. In brief, our application relied on MySQL to provide reliable storage, retrieval, and updating of user data. The database's configurability and scalability allowed our application to grow alongside user demands, with MySQL's optimized response time keeping our application running smoothly.

CHAPTER 3

IMPLEMENTATION AND RESULTS

3.1 IMPLEMENATAION:

```
public void jButton1ActionPerformed(ActionEvent ae){
    if(ae.getSource()==jButton1){
        String dob = ((JTextField) jDateChooser1.getDateEditor().getUiComponent()).getText();
        String firname = jTextField1.getText();
        String midname = jTextField2.getText();
        String lstname = jTextField3.getText();
       String age = jSpinner1.getValue().toString();
        char[] p1=jPasswordField1.getPassword();
       String pin = new String(value: p1);
        String gender = (String)jComboBox1.getSelectedItem();


    if(firname.isEmpty() || midname.isEmpty() || lstname.isEmpty() || dob.isEmpty() || age.isEmpty() || pin.isEmpty() || gender.isEmpty()){
JOptionPane.showMessageDialog(parentComponent: null, message: "Please fill in all fields.");
return; // to stop user from procedding forward AND TAKE ONLY CHARACTERS
}
    else if(!firname.matches(regex: "[a-zA-Z]+") || !midname.matches(regex: "[a-zA-Z]+") || !lstname.matches(regex: "[a-zA-Z]+")) {
JOptionPane.showMessageDialog(parentComponent: null, message: "Please enter only Characters in Name-Field ");
return; // to stop user from proceeding forward
}
    else if (pin.length() != 4) {
JOptionPane.showMessageDialog(parentComponent: null, message: "PIN should be exactly 4 digits.");
return; // to stop user from procedding forward
}


     Random random = new Random();
     cardnumber  =""+Math.abs(random.nextLong()%900000000L+50409360000L);
    jTextField4.setText(t: cardnumber);
```

```
    try {
        Statement statement = Conn.getStatement();
           String query = "INSERT INTO users (firstname, middlename, lastname, dob, age, pin, gender, cardnumber,balance) "
                   + "VALUES ('"+firname+"', '"+midname+"', '"+lstname+"', '"+dob+"', '"+age+"', '"+pin+"', '"+gender+"', '"+cardnumber+"',0)";
            statement.executeUpdate(string:query);

            JOptionPane.showMessageDialog(parentComponent: null, "Card number: "+ cardnumber + " \nsuccessfully saved.");
             System.out.println(x: "data sent");
}
    catch(Exception e){
       JOptionPane.showMessageDialog(parentComponent: null, message: "Error saving user information.");
       e.printStackTrace();
    }

    }
   this.dispose();
    main2 m1=new main2();
    m1.setVisible(b: true);
}
```

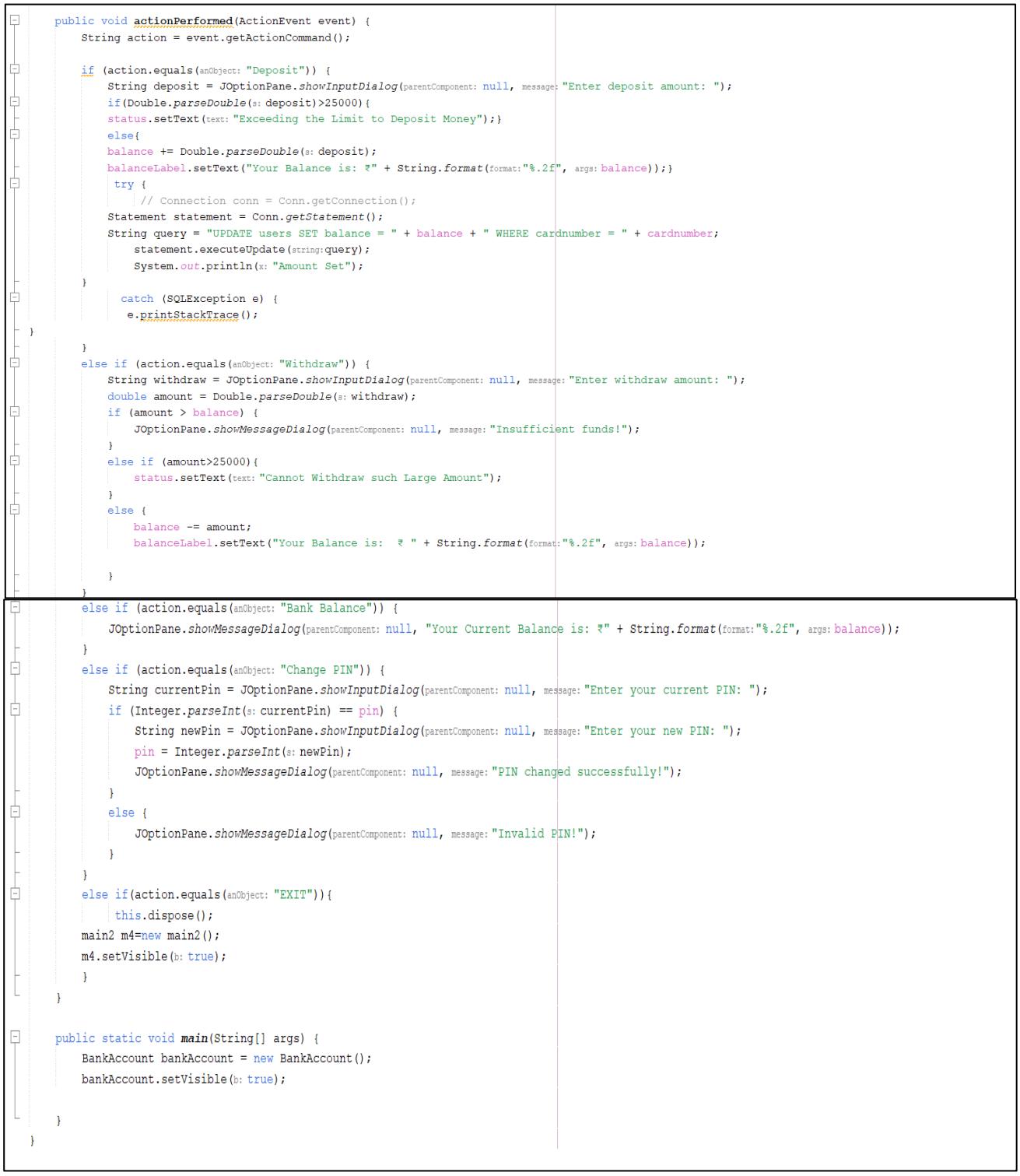
Fig 3: - Code to Take New User Data

```
    private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
        this.dispose();
        sign_up s1 = new sign_up();
       s1.setVisible(b: true);
    }


    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
      if(evt.getSource()==jButton2){
        cardnumber=jTextField1.getText();
        char[] p1=jPasswordField1.getPassword();
           String pin = new String(value: p1);
            try {
                // Connection conn = Conn.getConnection();
            Statement statement = Conn.getStatement();
            String query = "SELECT * FROM users WHERE cardnumber='" + cardnumber + "' AND pin='" + pin + "'";
            ResultSet rs = statement.executeQuery(string:query);

            if (rs.next()){
                // cardnumber and pin matched in sql, so login is successful
                JOptionPane.showMessageDialog(parentComponent: null, message: "Login successful!");

                BankAccount b1=new BankAccount();
                b1.setVisible(b: true);
                this.dispose();
            } else {
                // cardnumber and pin did not match the databse, so login failed
                JOptionPane.showMessageDialog(parentComponent: null, message: "Incorrect card number or pin. Please try again.");
                return;
            }
        }
            catch (Exception e) {
    e.printStackTrace();
}
```

Fig 3: -To Check the Inserted Data is Correct and let User Proceed further or Not

```
package atm_app;

import java.sql.*;

public class Conn {

    private static Connection connection;
    static Statement statement;

    public static Connection getConnection() {
        Connection Conn=null;
        return connection;
    }

    public static Statement getStatement() throws SQLException {
        statement = connection.createStatement();
        return statement;
    }
 public static ResultSet getResultset(String query) throws SQLException {
        statement = getStatement();
        ResultSet rs = statement.executeQuery(string:query);
        return rs;
    }
    static {
        try {
            Class.forName(className:"com.mysql.jdbc.Driver");
            connection = DriverManager.getConnection(url:"jdbc:mysql://localhost:3306/sign_up", user: "root", password: "MePushkar@sql#193?PW");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }



}
```

Fig 4: - MySQL Methods to make Connection with Java Classes using JDBC



```
    public void actionPerformed(ActionEvent event) {
        String action = event.getActionCommand();

        if (action.equals(anObject: "Deposit")) {
            String deposit = JOptionPane.showInputDialog(parentComponent: null, message: "Enter deposit amount: ");
            if(Double.parseDouble(s: deposit)>25000){
            status.setText(text: "Exceeding the Limit to Deposit Money");}
            else{
            balance += Double.parseDouble(s: deposit);
            balanceLabel.setText("Your Balance is: ₹" + String.format(format:"%.2f", args: balance));}
             try {
                 // Connection conn = Conn.getConnection();
            Statement statement = Conn.getStatement();
            String query = "UPDATE users SET balance = " + balance + " WHERE cardnumber = " + cardnumber;
                statement.executeUpdate(string:query);
                System.out.println(x: "Amount Set");
        }
              catch (SQLException e) {
               e.printStackTrace();
    }
        }
        else if (action.equals(anObject: "Withdraw")) {
            String withdraw = JOptionPane.showInputDialog(parentComponent: null, message: "Enter withdraw amount: ");
            double amount = Double.parseDouble(s: withdraw);
            if (amount > balance) {
                JOptionPane.showMessageDialog(parentComponent: null, message: "Insufficient funds!");
            }
            else if (amount>25000){
                status.setText(text: "Cannot Withdraw such Large Amount");
            }
            else {
                balance -= amount;
                balanceLabel.setText("Your Balance is:  ₹ " + String.format(format:"%.2f", args: balance));

            }
        }
        else if (action.equals(anObject: "Bank Balance")) {
            JOptionPane.showMessageDialog(parentComponent: null, "Your Current Balance is: ₹" + String.format(format:"%.2f", args: balance));
        }
        else if (action.equals(anObject: "Change PIN")) {
            String currentPin = JOptionPane.showInputDialog(parentComponent: null, message: "Enter your current PIN: ");
            if (Integer.parseInt(s: currentPin) == pin) {
                String newPin = JOptionPane.showInputDialog(parentComponent: null, message: "Enter your new PIN: ");
                pin = Integer.parseInt(s: newPin);
                JOptionPane.showMessageDialog(parentComponent: null, message: "PIN changed successfully!");
            }
            else {
                JOptionPane.showMessageDialog(parentComponent: null, message: "Invalid PIN!");
            }
        }
        else if(action.equals(anObject: "EXIT")){
             this.dispose();
        main2 m4=new main2();
        m4.setVisible(b: true);
        }
    }

    public static void main(String[] args) {
        BankAccount bankAccount = new BankAccount();
        bankAccount.setVisible(b: true);


    }
}
```

Fig 6: - To make Bank Transactions and various conditions to withdraw and deposit money

3.2 RESULTS:

Fig 7: - Main Page to Ask User for Login / Sign Up

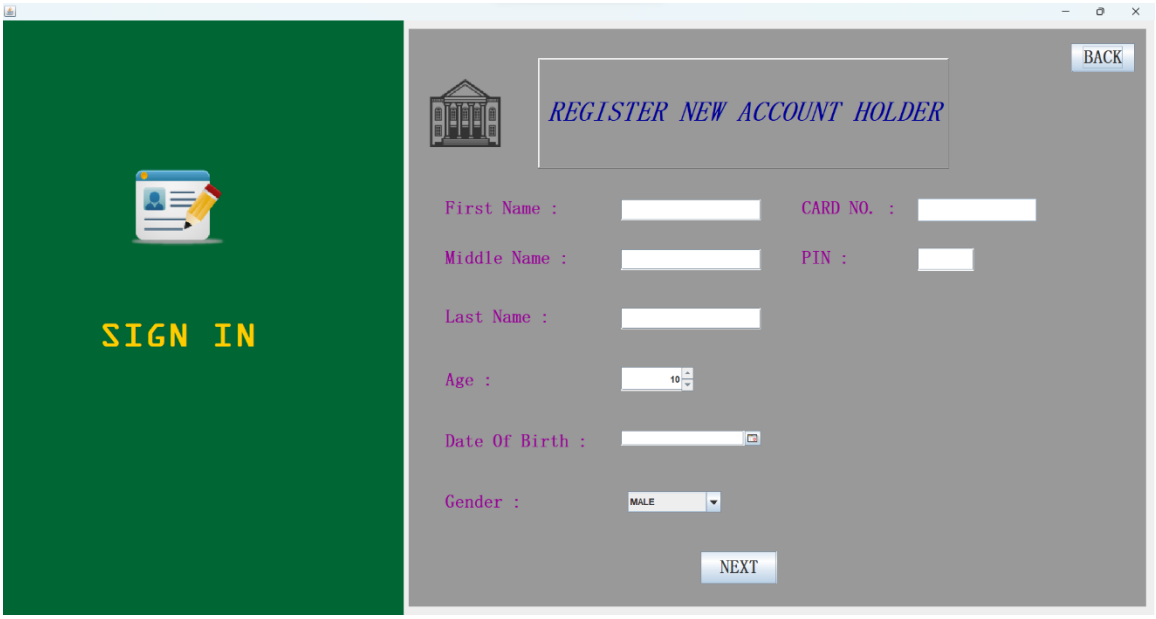


Fig 8: - Sign-In Page to Take New User Information

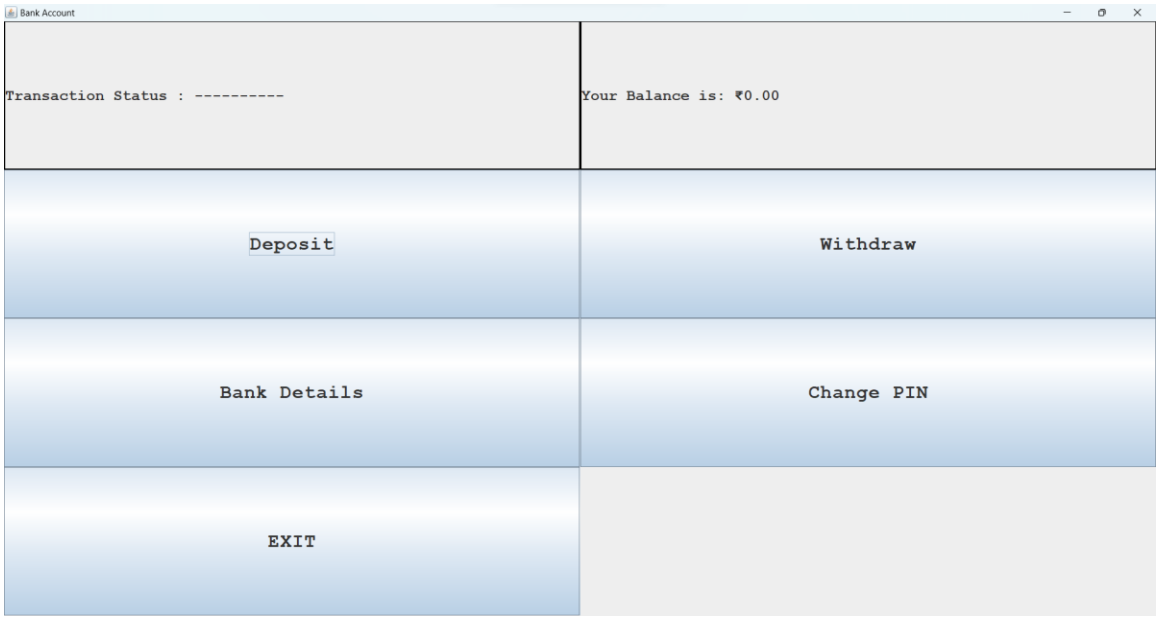


Fig 9: - Banking Transactions Option & Account Status

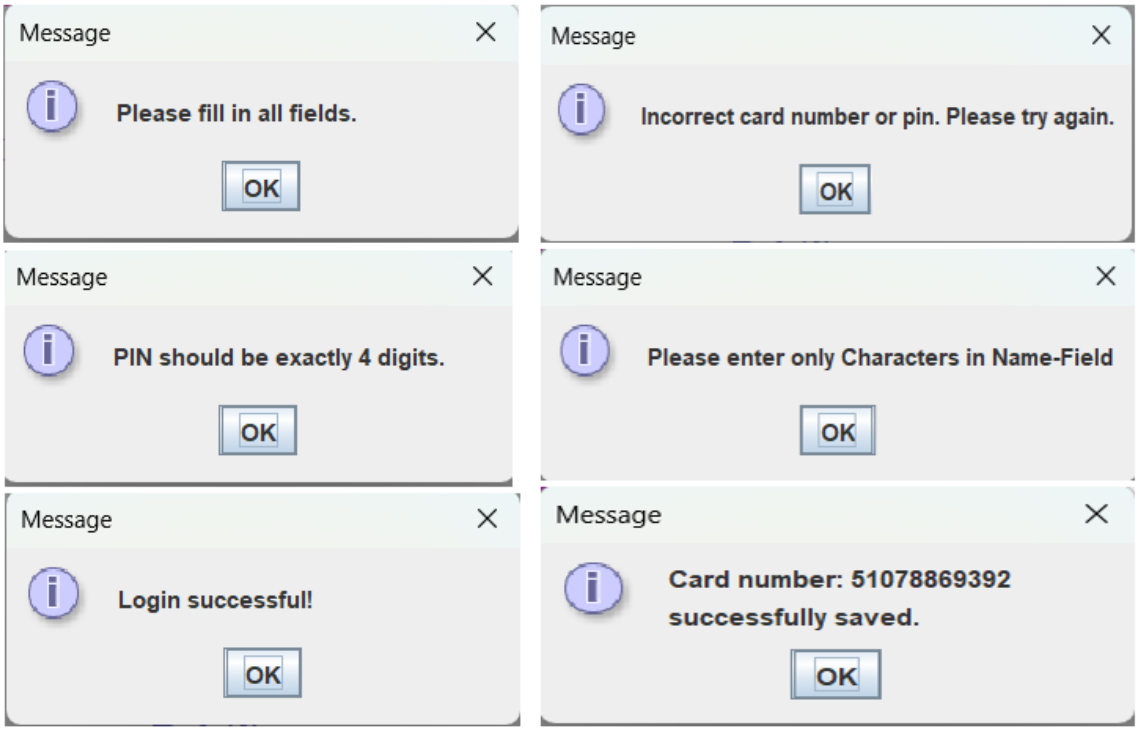


Fig 10: - To Show User Different Types of Messages and Warning

CHAPTER 4

CONCLUSION AND FUTURE SCOPE

4.1 Conclusion:

In this project, you learned how to design an ATM machine using Java. You started by creating classes for the ATM machine and bank accounts, implemented user authentication, provided menu options for various transactions, and handled actions based on the user's menu selection. Throughout the project, you also considered error handling and user input validation.

4.2 Future Scope:

Here are a few potential enhancements you can consider for your ATM machine project:

1. Implement support for multiple ATM machines and connect them to a centralized database for user account management.
2. Enhance security features, such as implementing two-factor authentication or using encryption to protect sensitive data.
3. Add features like fund transfers between different accounts, bill payments, or mobile recharge.
4. Improve the user interface by creating a graphical user interface (GUI) using frameworks like JavaFX or Swing.
5. Implement transaction history tracking to allow users to view their previous transactions.
6. Consider adding additional functionality, such as check deposit using image recognition or biometric authentication.

CHAPTER 5

REFERENCES

1. "Creating a Java ATM GUI Application", by Alex Coleman

**(https://www.codeproject.com/Articles/83235/Creating-a-Java-ATM-GUI-Application)

2. "Creating an ATM Application with Java Swing", by Sander DeVrieze

**(https://www.javacodegeeks.com/2015/09/creating-an-atm-application-with-java-swing.html)

3. "Java Swing ATM Tutorial", by Johnny Ernest
**(https://www.youtube.com/watch?v=A-geKfKFeJs)

4. "Java Programming Tutorial – ATM Machine Part-15 (ATM Project) in Java using Swing", by Easy Engineering Classes
**(https://www.youtube.com/watch?v=p8S0tLJCu4w)