Ex.No:11 MINI PROJECT

Aim:

The aim of the Bank Account Management System is to provide a user-friendly interface for managing bank accounts. This system allows users to create new accounts, deposit and withdraw funds, view transaction history, and edit account details.

Project Description:

Introduction:

The Bank Account Management System is designed to simplify and automate the management of individual bank accounts. It offers essential functionalities such as account creation, deposit, withdrawal, viewing transaction history, and editing account details. The system is implemented in Java and utilizes a graphical user interface (GUI) for a seamless user experience.

Scope of the Project:

The project encompasses the creation and management of individual bank accounts. Users can perform various transactions, and the system ensures the security and integrity of account information. It provides a convenient way for users to interact with their accounts and perform banking operations.

Mechanism Used:

1. Object-Oriented Design:

- Utilizes Java's object-oriented programming paradigm to create modular and extensible classes such as `BankAccount` and `PersonalFinanceManager`. This approach enhances code readability, reusability, and maintainability.

2. Graphical User Interface (GUI):

- Implements a GUI using AWT to provide users with an interactive and visually appealing experience. The GUI includes buttons and input fields for each functionality, enhancing the overall user interface.

Hardware and Software Requirements:

- Hardware:

-Personal Computer or Laptop

-Minimum 2 GB RAM

-Processor: Dual Core or higher

-Storage: 20 GB or more

- Software:

- Java Development Kit (JDK) for Java programming.

- An Integrated Development Environment (IDE) supporting Java (e.g., IntelliJ IDEA, Eclipse) for coding convenience.

- AWT library for building the graphical user interface and Swing libraries.

Algorithm:

Step1: Create Account:

-Collect user input for account name, security number, phone number, and address.

-Validate input, ensuring account name is at least 5 characters long and other details are provided.

-Create a new instance of the BankAccount class with the entered details.

-Add the account to the account list.

-Save the updated account list.

Step2:Deposit:

-Retrieve the current account.

-Collect user input for the deposit amount.

-Validate the deposit amount.

-Update the account balance.

-Save the updated account details.

- Update the account balance and transaction history.

Step4: Withdraw:

-Retrieve the current account.

-Collect user input for the withdrawal amount.

-Validate the withdrawal amount.

-Check for sufficient funds.

-Update the account balance.

-Record the withdrawal in the transaction history.

-Save the updated account details.

Step6: View Transaction History:

-Retrieve and display the transaction history for the selected account.

Step7:Edit Details Algorithm:

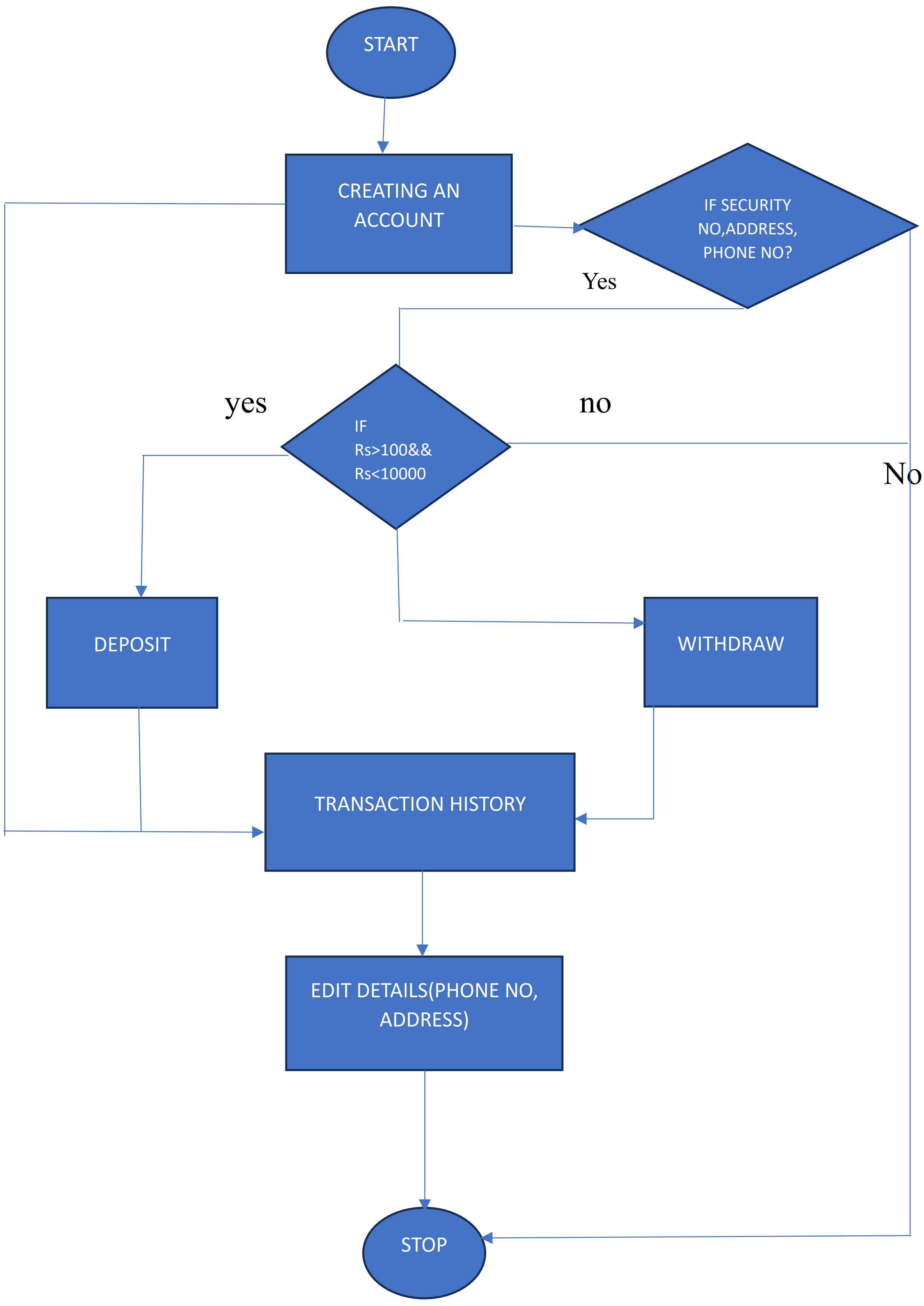
-Retrieve the current account.

-Collect and validate user inputs for new details.

-Update the account details.

-Save the updated details.

Flowchart:


START
CREATING AN ACCOUNT
IF SECURITY NO,ADDRESS, PHONE NO?
Yes
No
yes
no
IF Rs>100&& Rs<10000
DEPOSIT
WITHDRAW
TRANSACTION HISTORY
EDIT DETAILS(PHONE NO, ADDRESS)
STOP

Source Code:

```
import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.*;

import java.util.ArrayList;

import javax.swing.JOptionPane;

import java.util.List;

class BankAccount implements Serializable {

   private static final long serialVersionUID = 1L;

   private String accountName;

   private double balance;

   private String securityNumber;

   private String phoneNumber;

   private String address;

   private ArrayList<String> transactionHistory;

   public BankAccount(String accountName, String securityNumber, String phoneNumber,
String address,double balance) {

       this.accountName = accountName;

       this.balance = balance;

       this.securityNumber = securityNumber;

       this.phoneNumber = phoneNumber;

       this.address = address;

       this.transactionHistory = new ArrayList<>();}
```

```
}

public ArrayList<String> getTransactionHistory() {
        return transactionHistory;
    }
    public void addTransaction(String transaction) {
        if (transactionHistory == null) {
            transactionHistory = new ArrayList<>();
        }
        transactionHistory.add(transaction);
    }
    public void setTransactionHistory(ArrayList<String> transactionHistory) {
        this.transactionHistory = transactionHistory;
    }
    public String getAccountName() {
        return accountName;
    }
    public double getBalance() {
        return balance;
    }
    public String getSecurityNumber() {
        return securityNumber;
    }
    public String getPhoneNumber() {
        return phoneNumber;
    }

    public String getAddress() {
        return address;
    }

    public void deposit(double amount) {
        balance += amount;
        transactionHistory.add("Deposit: +" + amount);
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
            transactionHistory.add("Withdrawal: -" + amount);
        } else {
            System.out.println("Insufficient funds!");
        }
    }
}
```

```
class BankAccountGUI extends Frame implements ActionListener {
    private BankAccount currentAccount;
    private List<BankAccount> accountList;
private ArrayList<String> transactionHistory;
    private TextField accountNameField, amountField, recipientField, securityNumberField,
phoneNumberField, addressField;
    private TextArea outputArea;
    private TextField depositField;
    private TextField withdrawField,balanceField;
    public BankAccountGUI() {
       accountList = loadAccountList();
        currentAccount = null;
        Label accountNameLabel = new Label("Account Name:");
        Label amountLabel = new Label("Amount:");
        Label recipientLabel = new Label("Recipient:");
        Label securityNumberLabel = new Label("Security Number:");
        Label phoneNumberLabel = new Label("Phone Number:");
        Label addressLabel = new Label("Address:");
        accountNameField = new TextField(20);
        amountField = new TextField(20);
        recipientField = new TextField(20);
        securityNumberField = new TextField(20);
        phoneNumberField = new TextField(20);
        addressField = new TextField(20);
        Button createAccountButton = new Button("Create Account");
        Button depositButton = new Button("Deposit");
        Button withdrawButton = new Button("Withdraw");
        Button viewHistoryButton = new Button("View Transaction History");
        Button exitButton = new Button("Exit");
        Button resetButton = new Button("Reset");
        Button saveButton = new Button("Save Details");
        Button loadButton = new Button("Load Details");
        Button editButton = new Button("Edit");
        outputArea = new TextArea(20, 50);
    setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(5, 5, 5, 5);
    gbc.gridx = 0;
    gbc.gridy = 0;
    add(accountNameLabel, gbc);
    gbc.gridx = 1;
    gbc.gridy = 0;
    add(accountNameField, gbc);
    gbc.gridx = 3;
    gbc.gridy = 0;
    add(createAccountButton, gbc);
    gbc.gridx = 0;
    gbc.gridy = 1;
    add(amountLabel, gbc);
```

```
gbc.gridx = 1;

gbc.gridy = 1;

add(amountField, gbc);

gbc.gridx = 2;

gbc.gridy = 1;

add(depositButton, gbc);

gbc.gridx = 3;

gbc.gridy = 1;

add(withdrawButton, gbc);

gbc.gridx = 0;

gbc.gridy = 2;

add(recipientLabel, gbc);

gbc.gridx = 1;

gbc.gridy = 2;

add(recipientField, gbc);

gbc.gridx = 3;

gbc.gridy = 2;

add(viewHistoryButton, gbc);

gbc.gridx = 4;

gbc.gridy = 2;

add(exitButton, gbc);

gbc.gridx = 0;

gbc.gridy = 3;

add(securityNumberLabel, gbc);
```

```
gbc.gridx = 1;

gbc.gridy = 3;

add(securityNumberField, gbc);

gbc.gridx = 0;

gbc.gridy = 4;

add(phoneNumberLabel, gbc);

gbc.gridx = 1;

gbc.gridy = 4;

add(phoneNumberField, gbc);

gbc.gridx = 0;

gbc.gridy = 5;

add(addressLabel, gbc);

gbc.gridx = 1;

gbc.gridy = 5;

add(addressField, gbc);

gbc.gridx = 0;

gbc.gridy = 6;

gbc.gridwidth = 5;

gbc.fill = GridBagConstraints.BOTH;

add(outputArea, gbc);

gbc.gridx = 4;

gbc.gridy = 3;

add(resetButton, gbc);

gbc.gridx = 4;
gbc.gridy = 4;
add(saveButton, gbc);
gbc.gridx = 4;
gbc.gridy = 5;
add(loadButton, gbc);
gbc.gridx = 3;
gbc.gridy = 5;
add(editButton, gbc);
        createAccountButton.addActionListener(this);
        depositButton.addActionListener(this);
        withdrawButton.addActionListener(this);
        viewHistoryButton.addActionListener(this);
        exitButton.addActionListener(this);
        resetButton.addActionListener(this);
        saveButton.addActionListener(this);
        loadButton.addActionListener(this);
        editButton.addActionListener(this);
        setTitle("Bank Account Management System");
        setSize(600, 400);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
```

```
String command = e.getActionCommand();
switch (command) {
   case "Create Account":
      createAccount();
      break;
   case "Deposit":
      deposit();
      break;
   case "Withdraw":
      withdraw();
      break;
   case "View Transaction History":
      viewTransactionHistory();
      break;
   case "Reset":
      resetScreen();
      break;
   case "Save Details":
      saveDetails(getFilePath(currentAccount), currentAccount);
      break;
   case "Load Details":
      loadDetails();
      break;
      case "Edit":
```

```
                editDetails();

                break;

                case "Exit":

                System.exit(0);

                break;

            }

        }

private void loadDetails() {

    String accountName = accountNameField.getText();

    String filePath = "C:\\Users\\chris\\prog\\" + accountName + ".txt";

    try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {

   String line;

        String loadedAccountName = null;

        String loadedSecurityNumber = null;

        String loadedPhoneNumber = null;

        String loadedAddress = null;

        double loadedBalance = 0.0;

        ArrayList<String> loadedTransactionHistory = new ArrayList<>();

while ((line = reader.readLine()) != null) {

            if (line.startsWith("Account Name:")) {

                loadedAccountName = line.substring(line.indexOf("\"") + 1, line.lastIndexOf("\""));

            } else if (line.startsWith("Security Number:")) {

                loadedSecurityNumber          =          line.substring(line.indexOf("\"")          +          1,
line.lastIndexOf("\""));

            } else if (line.startsWith("Phone Number:")) {
```

```
            loadedPhoneNumber = line.substring(line.indexOf("\"") + 1, line.lastIndexOf("\""));
          } else if (line.startsWith("Address:")) {
            loadedAddress = line.substring(line.indexOf("\"") + 1, line.lastIndexOf("\""));
          } else if (line.startsWith("Balance:")) {
            String balanceStr = line.substring(line.indexOf("\"") + 1, line.lastIndexOf("\""));
            loadedBalance = Double.parseDouble(balanceStr);
          } else if (line.startsWith("Transaction History:")) {
            // Read transaction history until the end of the file
            while ((line = reader.readLine()) != null) {
              if (line.isEmpty()) {
                break;
              }
              loadedTransactionHistory.add(line.substring(line.indexOf("\"")            +            1,
line.lastIndexOf("\"")));
            }
          }
        }
        if   (loadedAccountName   !=   null   &&   loadedSecurityNumber   !=   null   &&
loadedPhoneNumber != null && loadedAddress != null) {
          currentAccount  =  new  BankAccount(loadedAccountName,  loadedSecurityNumber,
loadedPhoneNumber, loadedAddress, loadedBalance);
          currentAccount.setTransactionHistory(loadedTransactionHistory);

          outputArea.setText("Details     loaded     successfully!\nAccount     Name:     "     +
currentAccount.getAccountName() +
                "\nSecurity Number: " + currentAccount.getSecurityNumber() +
                "\nPhone Number: " + currentAccount.getPhoneNumber() +
                "\nAddress: " + currentAccount.getAddress() +
                "\nBalance: Rs." + currentAccount.getBalance()+"\n");
            displayTransactionHistory();
        } else {
            outputArea.setText("Error loading details from file. Invalid file format.");
        }
    } catch (IOException | NumberFormatException ex) {
        ex.printStackTrace();
        outputArea.setText("Error loading details from file.");
    }
}
private void displayTransactionHistory() {
    // Display transaction history...
    ArrayList<String> history = currentAccount.getTransactionHistory();
    StringBuilder transactionHistoryText = new StringBuilder("Transaction History:\n");

    for (String transaction : history) {
        transactionHistoryText.append(transaction).append("\n");
    }

    outputArea.append("\n"+transactionHistoryText.toString());
}
```

```
private void editDetails() {
    if (currentAccount != null) {
        String accountName = accountNameField.getText();
        String filePath = "C:\\Users\\chris\\prog\\" + accountName + ".txt";
        try (FileWriter fileWriter = new FileWriter(filePath, true);
             PrintWriter writer = new PrintWriter(new BufferedWriter(fileWriter))) {
            String newPhoneNumber = phoneNumberField.getText();
            if (!newPhoneNumber.isEmpty() && isValidPhoneNumber(newPhoneNumber)) {
                currentAccount.setPhoneNumber(newPhoneNumber);
                writer.println("Phone Number: \"" + newPhoneNumber + "\"");
            } else {
                outputArea.setText("Please provide a valid 10-digit phone number.");
                return; // Exit the method if phone number is invalid
            }
            String newAddress = addressField.getText();
            if (!newAddress.isEmpty()) {
                currentAccount.setAddress(newAddress);
                writer.println("Address: \"" + newAddress + "\"");
            }
            String depositAmountStr = amountField.getText();
            if (!depositAmountStr.isEmpty()) {
                double depositAmount = Double.parseDouble(depositAmountStr);
                currentAccount.deposit(depositAmount);
                writer.println("Balance: \"" + currentAccount.getBalance() + "\"");}
            writer.println("Transaction History:");

            for (String transaction : currentAccount.getTransactionHistory()) {

                writer.println("\"" + transaction + "\"");

            }

            outputArea.setText("Details updated and saved successfully!");

        } catch (IOException | NumberFormatException ex) {

            ex.printStackTrace();

            outputArea.setText("Error updating or saving details.");

        }

    } else {

        outputArea.setText("Please create or select an account first.");

    }

}

private void saveDetails(String filePath, BankAccount account) {

    try (PrintWriter writer = new PrintWriter(filePath)) {

        writer.println("Account Name: \"" + account.getAccountName() + "\"");

        writer.println("Balance: \"" + account.getBalance() + "\"");

        writer.println("Security Number: \"" + account.getSecurityNumber() + "\"");

        writer.println("Phone Number: \"" + account.getPhoneNumber() + "\"");

        writer.println("Address: \"" + account.getAddress() + "\"");

        writer.println("Transaction History:");


        // Save transaction history to file

        for (String transaction : account.getTransactionHistory()) {
            writer.println("\"" + transaction + "\"");
        }
        outputArea.setText("Details updated and saved successfully!");
    } catch (IOException ex) {
        ex.printStackTrace();
        outputArea.setText("Error saving details to file.");
    }
}
private BankAccount getAccountByName(String accountName) {


    return currentAccount.getAccountName().equals(accountName) ? currentAccount : null;
}
    private List<BankAccount> loadAccountList() {
        // Load the account list from a file (deserialize)
        try    (ObjectInputStream    ois    =    new    ObjectInputStream(new
FileInputStream("accountList.ser"))) {
            return (List<BankAccount>) ois.readObject();
        } catch (FileNotFoundException e) {
            // If the file is not found, return a new empty list
            return new ArrayList<>();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
            return new ArrayList<>();
        }
    }
    private void saveAccountList() {
        // Save the account list to a file (serialize)
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("accountList.ser"))) {
            oos.writeObject(accountList);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private boolean accountExists(String accountName) {
        for (BankAccount account : accountList) {
            if (account.getAccountName().equals(accountName)) {
                return true;
            }
        }
        return false;
    }
    private void createAccount() {
        // Your existing code...
        String accountName = accountNameField.getText();
        String securityNumber = securityNumberField.getText();
        String phoneNumber = phoneNumberField.getText();
        String address = addressField.getText();
        if (accountName.length() >= 5) {
```

```
        if (!securityNumber.isEmpty() && isValidPhoneNumber(phoneNumber) &&
!address.isEmpty()) {
          double initialBalance = 1000;
          currentAccount = new BankAccount(accountName, securityNumber, phoneNumber,
address, initialBalance);
          accountList.add(currentAccount); // Add the new account to the list
          saveAccountList(); // Save the updated list to the file
          outputArea.setText("Account created successfully!\nAccount Name: " +
currentAccount.getAccountName() +
              "\nSecurity Number: " + currentAccount.getSecurityNumber() +
              "\nPhone Number: " + currentAccount.getPhoneNumber() +
              "\nAddress: " + currentAccount.getAddress() +
              "\nBalance: Rs." + currentAccount.getBalance());
        } else {
          outputArea.setText("Please provide valid information...");
        }
      } else {
        outputArea.setText("Account name must have a minimum length of 5 characters.");
      }
    }
private boolean isValidPhoneNumber(String phoneNumber) {
  return phoneNumber.matches("\\d{10}");
}
private BankAccount loadAccountDetails(String accountName) {
  String filePath = getFilePath(accountName);
```

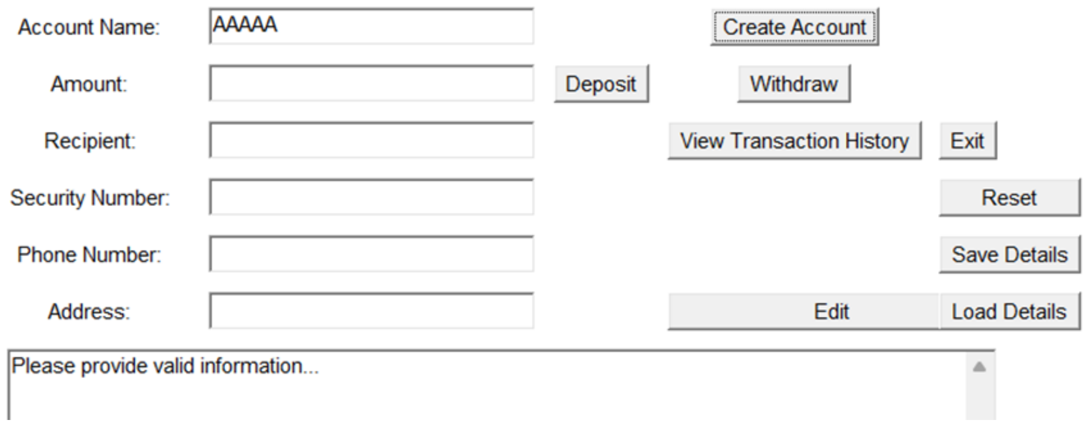
```
    try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
        String line;
        String loadedAccountName = null;
        String loadedSecurityNumber = null;
        String loadedPhoneNumber = null;
        String loadedAddress = null;
        double loadedBalance = 0.0;


        while ((line = reader.readLine()) != null) {
            if (line.startsWith("Account Name:")) {
                loadedAccountName = line.substring(line.indexOf("\"") + 1, line.lastIndexOf("\""));
            } else if (line.startsWith("Security Number:")) {
                loadedSecurityNumber = line.substring(line.indexOf("\"") + 1,
line.lastIndexOf("\""));
            } else if (line.startsWith("Phone Number:")) {
                loadedPhoneNumber = line.substring(line.indexOf("\"") + 1, line.lastIndexOf("\""));
            } else if (line.startsWith("Address:")) {
                loadedAddress = line.substring(line.indexOf("\"") + 1, line.lastIndexOf("\""));
            } else if (line.startsWith("Balance:")) {
                loadedBalance = Double.parseDouble(line.substring(line.indexOf("\"") + 1,
line.lastIndexOf("\"")));
            }
        }
        if (loadedAccountName != null && loadedSecurityNumber != null &&
loadedPhoneNumber != null && loadedAddress != null) {
```

```
            return new BankAccount(loadedAccountName, loadedSecurityNumber,
loadedPhoneNumber, loadedAddress, loadedBalance);
        } else {
            return null; // Invalid file format
        }
    } catch (IOException | NumberFormatException ex) {
        ex.printStackTrace();
        return null; // Error loading details from file
    }
}

// Helper method to get the file path for an account
private String getFilePath(String accountName) {
    return "C:\\Users\\chris\\prog\\" + accountName + ".txt";
}
private void resetScreen() {
    accountNameField.setText("");
    amountField.setText("");
    recipientField.setText("");
    securityNumberField.setText("");
    phoneNumberField.setText("");
    addressField.setText("");
    outputArea.setText("");

}
```

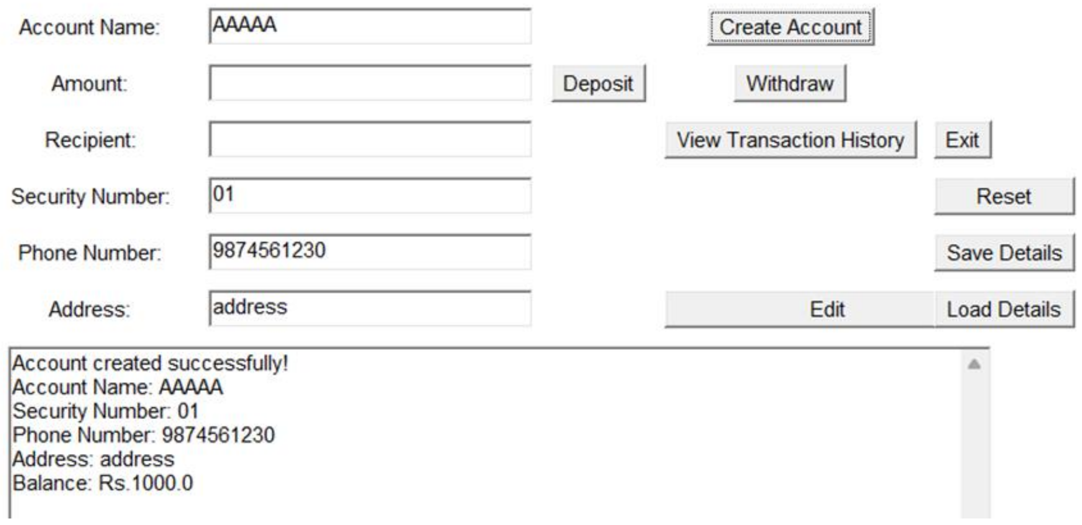
```
private String getFilePath(BankAccount account) {
    return "C:\\Users\\chris\\prog\\" + account.getAccountName() + ".txt";
}
private void deposit() {
    if (currentAccount != null) {
        double amount = Double.parseDouble(amountField.getText());
        if (amount >= 100) {
            currentAccount.deposit(amount);

            // Update details in the file
            saveDetails(getFilePath(currentAccount), currentAccount);

            outputArea.setText("Deposit successful!\nNew Balance: Rs." +
currentAccount.getBalance());
        } else {
            outputArea.setText("Deposit amount must be at least Rs.100.");
        }
    } else {
        outputArea.setText("Please create or select an account first.");
    }
}
private void withdraw() {
    if (currentAccount != null) {
        double amount = Double.parseDouble(amountField.getText());
        if (amount >= 100) {
```

```
currentAccount.withdraw(amount);
saveDetails(getFilePath(currentAccount), currentAccount);


         outputArea.setText("Withdrawal successful!\nNew Balance: Rs." + currentAccount.getBalance());
      } else {
         outputArea.setText("Withdrawal amount must be at least Rs.100.");
      }
   } else {
      outputArea.setText("Please create or select an account first.");
   }
}
private void viewTransactionHistory() {
   if (currentAccount != null) {
      // Display transaction history...
      ArrayList<String> history = currentAccount.getTransactionHistory();
      StringBuilder transactionHistoryText = new StringBuilder("Transaction History:\n");
      for (String transaction : history) {
         transactionHistoryText.append(transaction).append("\n");
      }
      outputArea.setText(transactionHistoryText.toString());
   } else {
      outputArea.setText("Please create or select an account first.");
   }
}}public class BankAccountManagementSystem {
    public static void main(String[] args) {

        new BankAccountGUI();

        List<BankAccount> accountList = new ArrayList<>();



    }

}
```

Output:

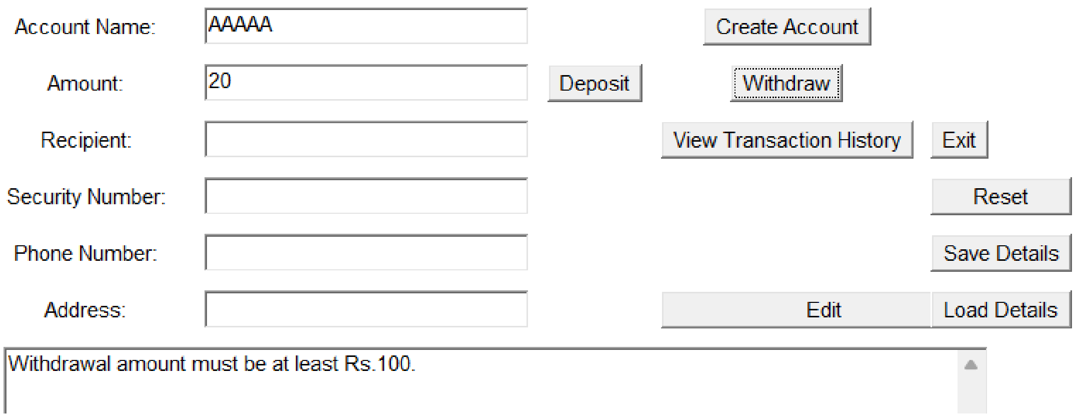


```
Please provide valid information...
```



```
Account created successfully!
Account Name: AAAAA
Security Number: 01
Phone Number: 9874561230
Address: address
Balance: Rs.1000.0
```

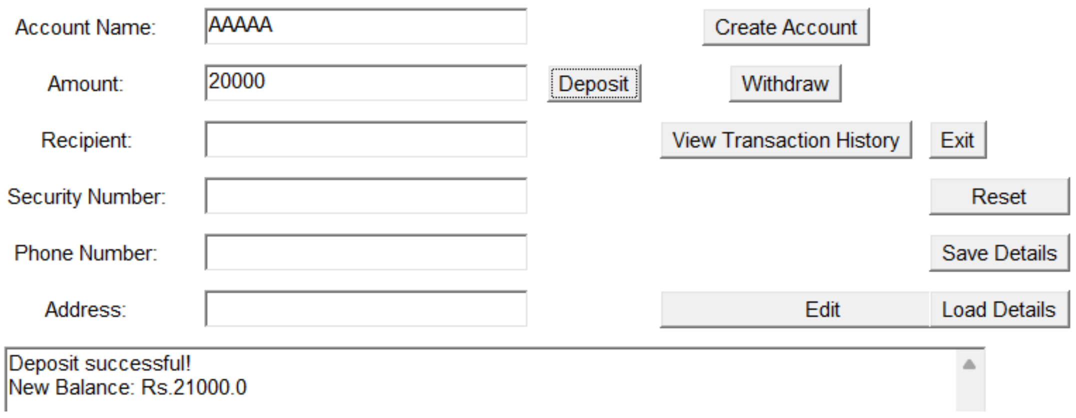
Figure 1: Output sample creating an account



Account Name:	AAAAA
Amount:	20

```
Withdrawal amount must be at least Rs.100.
```

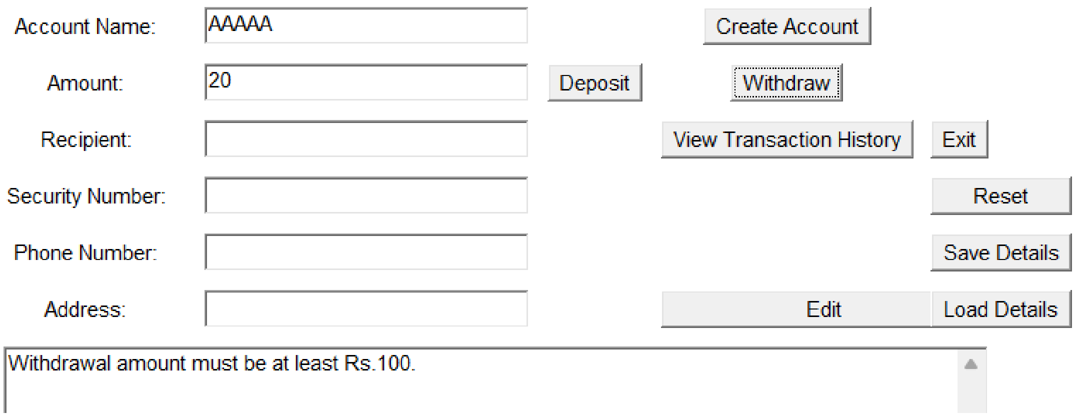
Figure2: Output sample with deposit amount=20



Account Name:	AAAAA
Amount:	20000

```
Deposit successful!
New Balance: Rs.21000.0
```

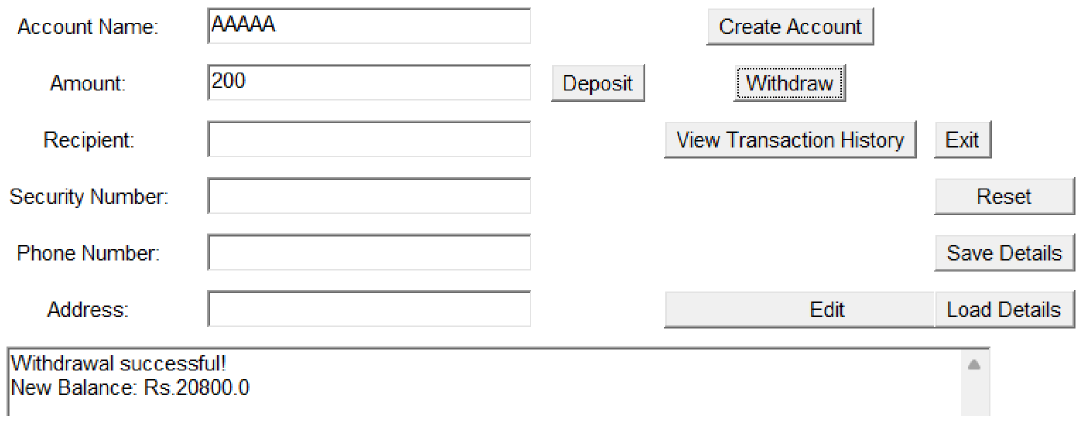
Figure 3: Output sample with deposit amount=20000



Account Name:	AAAAA
Amount:	20

```
Withdrawal amount must be at least Rs.100.
```

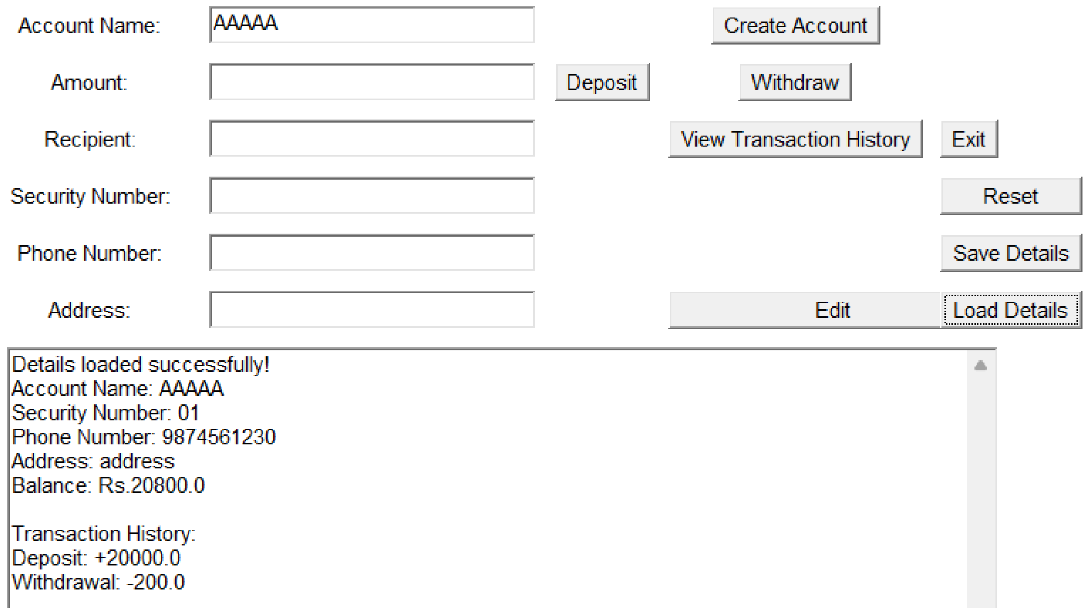
Figure 4: Output sample with withdrawal amount=20



Account Name: AAAAA
Amount: 200
Recipient:
Security Number:
Phone Number:
Address:

```
Withdrawal successful!
New Balance: Rs.20800.0
```

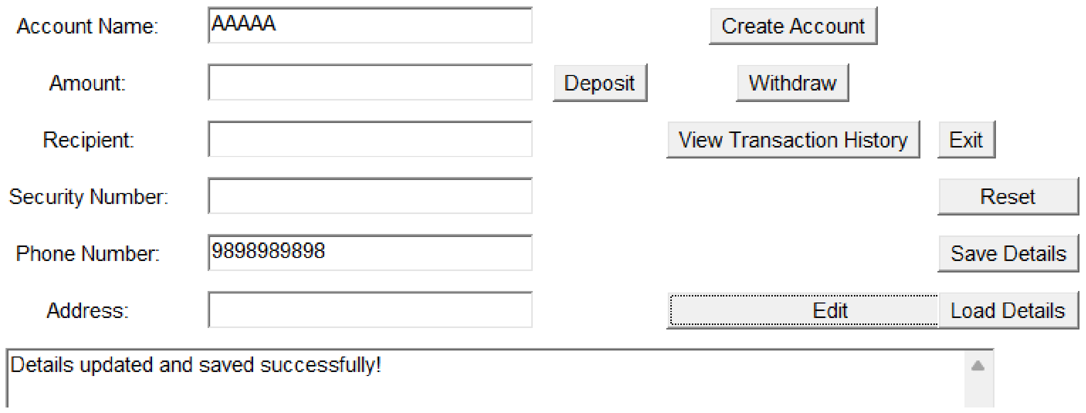
Figure 5: Output sample with withdrawal amount=200



Account Name: AAAAA
Amount:
Recipient:
Security Number:
Phone Number:
Address:

```
Details loaded successfully!
Account Name: AAAAA
Security Number: 01
Phone Number: 9874561230
Address: address
Balance: Rs.20800.0

Transaction History:
Deposit: +20000.0
Withdrawal: -200.0
```

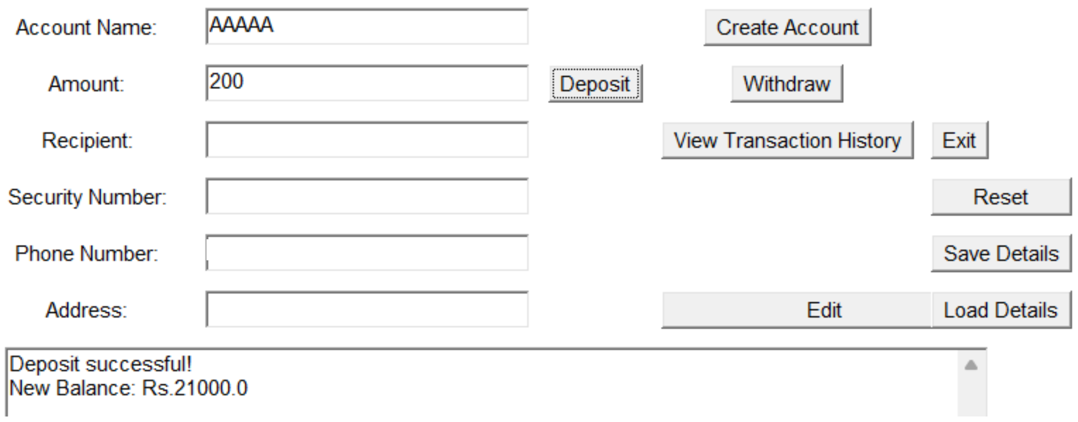
Figure 6: Output sample Loading the account details after exiting the program from a text file



Account Name: AAAAA
Amount:
Recipient:
Security Number:
Phone Number: 9898989898
Address:

```
Details updated and saved successfully!
```

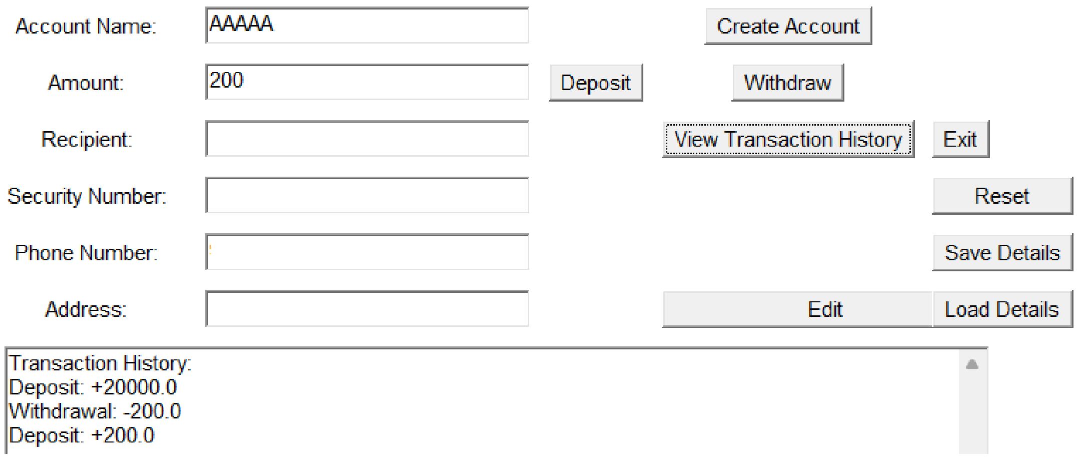
Figure 7: Output sample Editing the account details(Only phone no and address)



Account Name: AAAAA
Amount: 200
Recipient:
Security Number:
Phone Number:
Address:

```
Deposit successful!
New Balance: Rs.21000.0
```

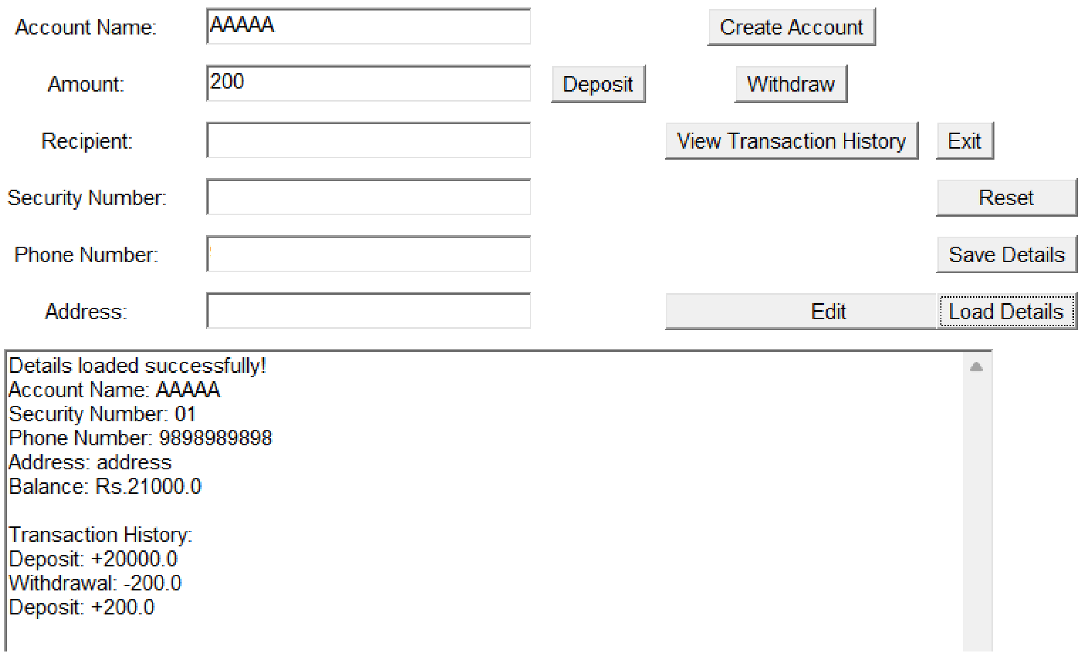
Figure 8: Output sample Depositing 200 after loading from text file



Account Name: AAAAA
Amount: 200
Recipient:
Security Number:
Phone Number:
Address:

```
Transaction History:
Deposit: +20000.0
Withdrawal: -200.0
Deposit: +200.0
```

Figure 9: Output sample transaction history



Account Name: AAAAA
Amount: 200
Recipient:
Security Number:
Phone Number:
Address:

```
Details loaded successfully!
Account Name: AAAAA
Security Number: 01
Phone Number: 9898989898
Address: address
Balance: Rs.21000.0

Transaction History:
Deposit: +20000.0
Withdrawal: -200.0
Deposit: +200.0
```

Figure 10: Output sample loading details

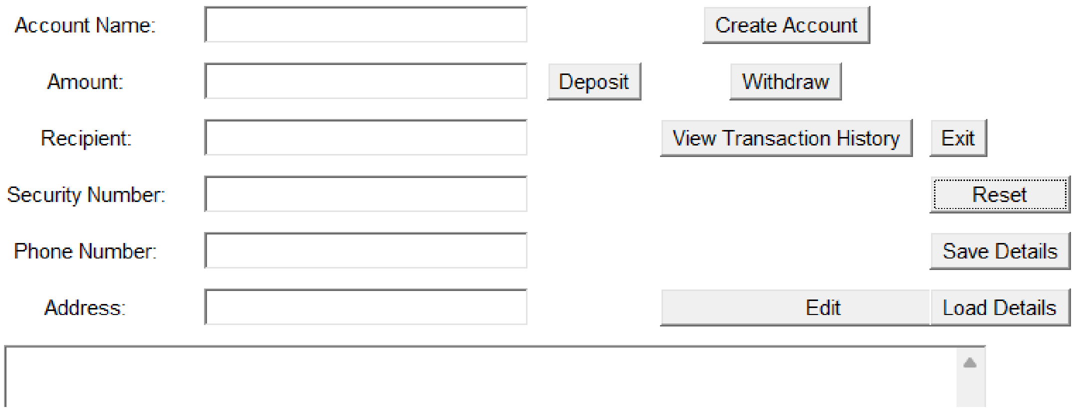


Figure 11: Output sample reset to clear previous account details in text box

<u>Result:</u>

The Bank Account Management System provides a user-friendly interface for effective management of bank accounts. Users can perform various operations seamlessly, and the system ensures data integrity by persistently storing account information. The graphical interface enhances user experience, making banking operations more accessible and efficient.