**FULL CODE**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.sql.\*;

// Model class to hold account details

class AccountBank {

    int accountNumber;

    String accountHolderName;

    double balance;

    int cibilScore;

    public AccountBank(int accountNumber, String accountHolderName, double balance, int cibilScore) {

        this.accountNumber = accountNumber;

        this.accountHolderName = accountHolderName;

        this.balance = balance;

        this.cibilScore = cibilScore;

    }

}

// Database management class

class BankManagementSystem {

    private Connection conn;

    public BankManagementSystem() {

        try {

            // Establish SQLite connection

            conn = DriverManager.getConnection("jdbc:sqlite:bank\_management.db");

            createTableIfNotExists();

        } catch (SQLException e) {

            JOptionPane.showMessageDialog(null, "Database connection error: " + e.getMessage());

        }

    }

    private void createTableIfNotExists() throws SQLException {

        String createTableSQL = "CREATE TABLE IF NOT EXISTS accounts (" +

                                "accountNumber INTEGER PRIMARY KEY, " +

                                "accountHolderName TEXT NOT NULL, " +

                                "balance REAL NOT NULL, " +

                                "cibilScore INTEGER NOT NULL" +

                                ");";

        try (Statement stmt = conn.createStatement()) {

            stmt.execute(createTableSQL);

        }

    }

    public void createAccount(int accountNumber, String accountHolderName, double initialDeposit) {

        if (findAccount(accountNumber) != null) {

            JOptionPane.showMessageDialog(null, "Account with account number " + accountNumber + " already exists.");

            return;

        }

        String insertSQL = "INSERT INTO accounts (accountNumber, accountHolderName, balance, cibilScore) VALUES (?, ?, ?, ?)";

        try (PreparedStatement pstmt = conn.prepareStatement(insertSQL)) {

            pstmt.setInt(1, accountNumber);

            pstmt.setString(2, accountHolderName);

            pstmt.setDouble(3, initialDeposit);

            pstmt.setInt(4, 700); // Default CIBIL score

            pstmt.executeUpdate();

            JOptionPane.showMessageDialog(null, "Account created successfully for " + accountHolderName);

        } catch (SQLException e) {

            JOptionPane.showMessageDialog(null, "Error creating account: " + e.getMessage());

        }

    }

    public void displayAllAccounts() {

        String querySQL = "SELECT \* FROM accounts";

        try (Statement stmt = conn.createStatement(); ResultSet rs = stmt.executeQuery(querySQL)) {

            StringBuilder accountsInfo = new StringBuilder();

            while (rs.next()) {

                accountsInfo.append("Account Number: ").append(rs.getInt("accountNumber")).append("\n")

                            .append("Account Holder: ").append(rs.getString("accountHolderName")).append("\n")

                            .append("Balance: ").append(rs.getDouble("balance")).append("\n")

                            .append("CIBIL Score: ").append(rs.getInt("cibilScore")).append("\n\n");

            }

            JOptionPane.showMessageDialog(null, accountsInfo.length() == 0 ? "No accounts to display." : accountsInfo.toString());

        } catch (SQLException e) {

            JOptionPane.showMessageDialog(null, "Error displaying accounts: " + e.getMessage());

        }

    }

    public void deposit(int accountNumber, double amount) {

        AccountBank account = findAccount(accountNumber);

        if (account != null) {

            double newBalance = account.balance + amount;

            int newCibilScore = account.cibilScore + (amount > 100000 ? 10 : 5);

            String updateSQL = "UPDATE accounts SET balance = ?, cibilScore = ? WHERE accountNumber = ?";

            try (PreparedStatement pstmt = conn.prepareStatement(updateSQL)) {

                pstmt.setDouble(1, newBalance);

                pstmt.setInt(2, newCibilScore);

                pstmt.setInt(3, accountNumber);

                pstmt.executeUpdate();

                JOptionPane.showMessageDialog(null, "Amount deposited successfully!");

            } catch (SQLException e) {

                JOptionPane.showMessageDialog(null, "Error depositing amount: " + e.getMessage());

            }

        } else {

            JOptionPane.showMessageDialog(null, "Account not found.");

        }

    }

    public void withdraw(int accountNumber, double amount) {

        AccountBank account = findAccount(accountNumber);

        if (account != null) {

            if (account.balance >= amount) {

                double newBalance = account.balance - amount;

                int newCibilScore = account.cibilScore - 5;

                String updateSQL = "UPDATE accounts SET balance = ?, cibilScore = ? WHERE accountNumber = ?";

                try (PreparedStatement pstmt = conn.prepareStatement(updateSQL)) {

                    pstmt.setDouble(1, newBalance);

                    pstmt.setInt(2, newCibilScore);

                    pstmt.setInt(3, accountNumber);

                    pstmt.executeUpdate();

                    JOptionPane.showMessageDialog(null, "Amount withdrawn successfully!");

                } catch (SQLException e) {

                    JOptionPane.showMessageDialog(null, "Error withdrawing amount: " + e.getMessage());

                }

            } else {

                JOptionPane.showMessageDialog(null, "Insufficient balance.");

            }

        } else {

            JOptionPane.showMessageDialog(null, "Account not found.");

        }

    }

    public void displayCibilScore(int accountNumber) {

        AccountBank account = findAccount(accountNumber);

        if (account != null) {

            JOptionPane.showMessageDialog(null, "CIBIL Score: " + account.cibilScore);

        } else {

            JOptionPane.showMessageDialog(null, "Account not found.");

        }

    }

    private AccountBank findAccount(int accountNumber) {

        String querySQL = "SELECT \* FROM accounts WHERE accountNumber = ?";

        try (PreparedStatement pstmt = conn.prepareStatement(querySQL)) {

            pstmt.setInt(1, accountNumber);

            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {

                return new AccountBank(rs.getInt("accountNumber"), rs.getString("accountHolderName"),

                                       rs.getDouble("balance"), rs.getInt("cibilScore"));

            }

        } catch (SQLException e) {

            JOptionPane.showMessageDialog(null, "Error finding account: " + e.getMessage());

        }

        return null;

    }

}

// Main class for GUI and interaction

public class Main extends Frame implements ActionListener {

    BankManagementSystem bank = new BankManagementSystem();

    Button createAccountButton, displayAccountsButton, depositButton, withdrawButton, cibilScoreButton;

    public Main() {

        setLayout(new FlowLayout());

        setBackground(Color.PINK); // Set background color to pink

        // Window listener to close the application when the window is closed

        addWindowListener(new WindowAdapter() {

            public void windowClosing(WindowEvent we) {

                System.exit(0);

            }

        });

        // Create Account Button

        createAccountButton = new Button("Create Account");

        createAccountButton.addActionListener(this);

        add(createAccountButton);

        // Display All Accounts Button

        displayAccountsButton = new Button("Display All Accounts");

        displayAccountsButton.addActionListener(this);

        add(displayAccountsButton);

        // Deposit Button

        depositButton = new Button("Deposit");

        depositButton.addActionListener(this);

        add(depositButton);

        // Withdraw Button

        withdrawButton = new Button("Withdraw");

        withdrawButton.addActionListener(this);

        add(withdrawButton);

        // Check CIBIL Score Button

        cibilScoreButton = new Button("Check CIBIL Score");

        cibilScoreButton.addActionListener(this);

        add(cibilScoreButton);

        // Set Frame properties

        setSize(400, 400);

        setTitle("Bank Management System");

        setVisible(true);

    }

    // Handling button actions

    public void actionPerformed(ActionEvent e) {

        try {

            if (e.getSource() == createAccountButton) {

                String accNumInput = JOptionPane.showInputDialog(this, "Enter Account Number:");

                if (accNumInput == null) return; // Exit if cancelled

                int accountNumber = Integer.parseInt(accNumInput);

                String accountHolder = JOptionPane.showInputDialog(this, "Enter Account Holder Name:");

                if (accountHolder == null) return;

                String initialDepositInput = JOptionPane.showInputDialog(this, "Enter Initial Deposit:");

                if (initialDepositInput == null) return;

                double initialDeposit = Double.parseDouble(initialDepositInput);

                bank.createAccount(accountNumber, accountHolder, initialDeposit);

            } else if (e.getSource() == displayAccountsButton) {

                bank.displayAllAccounts();

            } else if (e.getSource() == depositButton) {

                String accNumInput = JOptionPane.showInputDialog(this, "Enter Account Number:");

                if (accNumInput == null) return;

                int accountNumber = Integer.parseInt(accNumInput);

                String amountInput = JOptionPane.showInputDialog(this, "Enter Deposit Amount:");

                if (amountInput == null) return;

                double amount = Double.parseDouble(amountInput);

                bank.deposit(accountNumber, amount);

            } else if (e.getSource() == withdrawButton) {

                String accNumInput = JOptionPane.showInputDialog(this, "Enter Account Number:");

                if (accNumInput == null) return;

                int accountNumber = Integer.parseInt(accNumInput);

                String amountInput = JOptionPane.showInputDialog(this, "Enter Withdrawal Amount:");

                if (amountInput == null) return;

                double amount = Double.parseDouble(amountInput);

                bank.withdraw(accountNumber, amount);

            } else if (e.getSource() == cibilScoreButton) {

                String accNumInput = JOptionPane.showInputDialog(this, "Enter Account Number:");

                if (accNumInput == null) return;

                int accountNumber = Integer.parseInt(accNumInput);

                bank.displayCibilScore(accountNumber);

            }

        } catch (NumberFormatException ex) {

            JOptionPane.showMessageDialog(this, "Invalid input. Please enter valid numbers.");

        } catch (Exception ex) {

            JOptionPane.showMessageDialog(this, "Operation cancelled or an error occurred: " + ex.getMessage());

        }

    }

    public static void main(String[] args) {

        new Main();

    }

}