CORE JAVA PROJECT

***A JAVA-BASED ATM INTERFACE WITH JDBC CONNECTION***

**AIM:**

To achieve an ATM Interface simulation using JAVA and MySQL database.

**ABSTRACT:**

The ATM Interface System is a software application that emulates the functionality of an Automated Teller Machine (ATM) through a user-friendly digital interface. This project encompasses key banking operations, including account creation, balance inquiries, deposits, withdrawals, and transaction history retrieval. Users can securely register accounts, set personal identification numbers (PINs), and perform various transactions.

The system offers a secure and efficient means for account holders to manage their finances remotely. It ensures that users can easily access their accounts, check their balances, deposit funds, withdraw cash, and view transaction records within specified date ranges. The interface is designed to replicate the intuitive and familiar experience of using a physical ATM, facilitating ease of use and providing a sense of security to users.

Overall, the ATM Interface System aims to bring the convenience and accessibility of traditional ATM services to the digital realm, meeting the evolving needs of modern banking while emphasizing user-friendliness and security.

**SYNOPSIS:**

The ATM Interface System is a comprehensive software application designed to simulate an Automated Teller Machine (ATM) experience, providing a range of banking operations to users. The key features of this project are:

* Account Management: Users can open a new account by providing necessary details and set a 4-digit PIN for secure access.
* Balance Inquiry: Account holders can check their account balance through this system, ensuring efficient monitoring of their finances.
* Deposit Functionality: Users can deposit funds into their accounts, enhancing account liquidity and facilitating savings.
* Withdrawal Capability: Account holders can withdraw funds from their accounts, ensuring easy access to their money.
* Transaction History: The system allows users to view a customized transaction statement within a specified date range, aiding in tracking their financial activity.

The project aims to replicate real-world ATM functionalities within a digital interface, providing a user-friendly experience. It ensures secure account access and offers fundamental banking operations such as balance inquiry, deposits, and withdrawals. Additionally, the system allows users to review their transaction history, enhancing transparency and financial management. Overall, the ATM Interface System seeks to provide a convenient and secure means for users to conduct essential banking transactions digitally.

**SOFTWARE REQUIREMENTS:**

* **Operating System:** Windows 10 or later
* **Java Development Kit (JDK):** JDK 8 or later for Java-based development
* **Integrated Development Environment (IDE):** IntelliJ IDEA or Eclipse or NetBeans
* **Database Management System:** MySQL
* **Database Connectivity:** JDBC (Java Database Connectivity) library

**INITIAL SETUP:**

**Java Development Kit (JDK) Installation:**

* Download and install the latest version of JDK compatible with your operating system.
* Set the JAVA\_HOME environment variable to the JDK installation directory.

**Integrated Development Environment (IDE) Setup:**

* Choose and install an IDE such as IntelliJ IDEA, Eclipse, or NetBeans.
* Configure the IDE to use the installed JDK.

**Database Setup:**

* Install MySQL database server on your machine or use an existing server.
* Create a database schema for the ATM Interface System with two tables for Account and Transaction.

**Database Connectivity Setup:**

* Download the JDBC driver for your chosen database and add it to your project.

**Dependency Management:**

* If using Maven, ensure the build files (pom.xml for Maven) specify project dependencies correctly.

**Configuration:**

* Configure the database connection parameters (e.g., database URL, username, password) in the project's configuration files.

**Build and Run:**

* Build the project using the IDE or build automation tool.
* Run the application, ensuring that the database connection is established and functional.

**PROJECT CODE:**

**ATM MAIN:**

The ATMMain class is the entry point and orchestrator of the ATM Interface System. Here's a brief explanation of its working:

* **Initialization:**

Creates instances of AtmOperations and TransactionStatement for handling ATM operations and transactions.

* **Welcome Message and Menu Display:**

Displays a welcome message and presents a menu of available operations to the user.

* **User Input and Operation Selection:**

Prompts the user to enter their desired operation (e.g., Check Balance, Open an Account, Deposit, Withdraw, Generate Mini Statement)

* **Operation Execution:**

Based on the user's input, it invokes appropriate methods from AtmOperations or TransactionStatement to perform the selected operation.

* **Continuation or Exit:**

Asks the user if they want to perform more transactions ('y' to continue) or exit ('n' to exit). If the user chooses to exit, the program terminates.

The class encapsulates the core logic for user interaction, operation execution, and program flow control in the ATM system. It utilizes the Scanner class to read user input and directs the flow of the program based on the user's choices.

The methods in AtmOperations and TransactionStatement perform the actual ATM operations, such as checking balance, registering accounts, depositing, withdrawing, and generating mini statements, while this main class handles the overall flow and coordination of these operations based on user input.

AtmMain:

package com.atmproject;

import java.sql.SQLException;

import java.util.Scanner;

/\*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-Core Java Project-\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ATM Interface Project\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*under the guidance of Trainer Indrakka Basappa Mali

\*

\* This is the main class for the ATM Interface System.

\* It contains the main method to start the ATM application and handle user interactions.

\*

\* @author Surendharan

\* @version 1.0

\*/

public class AtmMain {

public static void main(String[] args) throws SQLException {

AtmOperations atmMenu = new AtmOperations();

TransactionStatement transaction = new TransactionStatement();

Scanner sc = new Scanner(System.in);

System.out.println("\*\*\*\*\*\*\*\*\*\*Welcome to ABC Bank!\*\*\*\*\*\*\*\*\*\*");

System.out.println("\nPlease choose the options below to proceed with the transaction");

//Get the choice from the user

while(true) {

System.out.println("\n==========MENU==========");

System.out.println("1.Check Balance");

System.out.println("2.Open an account");

System.out.println("3.Deposit Money");

System.out.println("4.Withdraw Money");

System.out.println("5.Generate Mini Statement");

System.out.print("\nEnter the option that you want to proceed: ");

int option = sc.nextInt();

switch(option) {

case 1:

atmMenu.checkBalance();

break;

case 2:

atmMenu.register();

break;

case 3:

atmMenu.deposit();

break;

case 4:

atmMenu.withdraw();

break;

case 5:

transaction.getCustomTransactionStatement();

break;

default:

//If the user enters the option other than mentioned in the menu the message will be displayed

System.out.println("Please enter an option from the menu");

continue;

}

System.out.print("\nPlease enter 'y' to continue for other transactions or 'n' to exit :");

char lastOption = sc.next().toLowerCase().charAt(0);

if(lastOption != 'y') {

System.out.println("\*\*\*\*Thank you for using our services!\*\*\*\*");

break;

}

}

sc.close();

}

}

**DATABASE CONNECTION CLASS:**

* The database connection is established using the driver class of the SQL and connecting it with the localhost.
* The database name is ‘atm\_db’ and is connected using this statement in the class.

private static String url = "jdbc:mysql://localhost:3306/atm\_db";

package com.atmproject;

import java.sql.Connection;

import java.sql.DriverManager;

public class DatabaseConnection {

//Initializing Driver class and setting up the url

private static String driver = "com.mysql.cj.jdbc.Driver";

private static String url = "jdbc:mysql://localhost:3306/atm\_db";

private static String uname = "root";

private static String pass = "root";

private static Connection conn;

public DatabaseConnection() {}

public static Connection getConnection() {

try {

Class.forName(driver);

if(conn == null) {

//Make connection with the SQL database

conn = DriverManager.getConnection(url, uname, pass);

}

} catch (Exception e) {

e.printStackTrace();

}

return conn;

}

}

**ATM OPERATIONS CLASS:**

The AtmOperations class in the ATM Interface System contains methods to perform various ATM operations. Here's a brief explanation of its workings:

* **Account Number and PIN Validation:**

inputandvalidateAccountNo() and inputandvalidatePin() methods validate the entered account number and PIN to ensure they meet the required criteria.

* **Amount Validation:**

inputandvalidateAmount() method validates the amount entered for transactions, ensuring it is a valid amount.

* **Mobile Number Validation:**

inputandvalidateMobileNo() method validates the entered mobile number to meet the specified criteria.

* **Account Existence Check:**

hasAccNoInDatabase() method checks if an account number exists in the database.

* **PIN Verification:**

hasPinInAccount() method verifies if the PIN provided matches the stored PIN for a given account.

* **Balance Checking:**

checkBalance() method allows users to check their account balance.

* **Account Registration:**

register() method facilitates the registration of a new account.

* **Deposit and Withdrawal:**

deposit() and withdraw() methods handle the deposit and withdrawal transactions, respectively.

* **Helper Method for Balance Retrieval:**

getBalance() method retrieves the account balance.

The class provides a centralized location for implementing various ATM operations, ensuring input validation, account and PIN checks, and facilitating interactions with the database. The methods are designed to enhance reusability, modularity, and maintainability in the ATM Interface System.

AtmOperations:

package com.atmproject;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.Scanner;

public class AtmOperations {

private static Connection con;

private static Statement st;

private static PreparedStatement pst;

private static PreparedStatement pst1;

private static PreparedStatement pst2;

private static ResultSet rs;

private static ResultSet rs1;

private static ResultSet rs2;

//This method will get and validate the Account Number

static String inputandvalidateAccountNo() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the Account Number: ");

String accNo = sc.next();

if (accNo.length() != 12) {

System.err.println("Account no is not valid. Please check and enter correct Account number");

return inputandvalidateAccountNo();

}

else {

return accNo;

}

}

//This method will get and validate the Account Pin

static String inputandvalidatePin() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter your 4-digit PIN: ");

String pin = sc.next();

if (pin.length() != 4) {

System.err.println("PIN is not valid. Please check and enter correct PIN");

return inputandvalidatePin();

}

else {

return pin;

}

}

//This method will get and validate the Amount for withdraw and Deposit

private static double inputandvalidateAmount() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the Amount: ");

int amount = sc.nextInt();

if(amount == 0) {

System.err.println("Please enter a valid amount");

return inputandvalidateAmount();

}

else if(amount < 100){ //Minimum Balance is Rs. 100

System.err.println("Enter a amount equal to or greater than Rs.100");

return inputandvalidateAmount();

}

else {

return amount;

}

}

//For getting and validating Mobile no for creating account

private static String inputandvalidateMobileNo() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter your Mobile Number: ");

String mobNo = sc.next();

// Check length and first digit

if (mobNo.length() != 10 || "12345".indexOf(mobNo.charAt(0)) != -1) {

System.err.println("Mobile number is not valid.");

System.out.println("Please check the mobile number is of 10-digit and not start with '1,2,3,4 or 5'");

return inputandvalidateMobileNo();

} else {

return mobNo;

}

}

//This method will verify whether the account number exists in the Bank Account database

private static boolean hasAccNoInDatabase(String accNo) throws SQLException {

String inquery = "SELECT account\_number FROM Account";

con = DatabaseConnection.getConnection();

st = con.createStatement();

rs = st.executeQuery(inquery);

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

// Store account numbers from the result set

while (rs.next()) {

String accno = rs.getString("account\_number");

accNoList.add(accno);

}

// Check if accNo exists in the list of account numbers

return accNoList.contains(accNo);

}

//This method will verify whether the PIN exists in the Bank Account database

private static boolean hasPinInAccount(String pin, String accNo) throws SQLException {

String query = "SELECT account\_pin FROM Account WHERE account\_number = " + accNo;

con = DatabaseConnection.getConnection();

st = con.createStatement();

rs = st.executeQuery(query);

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

// Store account PINs from the result set

while (rs.next()) {

String accountPin = rs.getString("account\_pin");

pinList.add(accountPin);

}

// Check if the pin exists in the list of account PINs

return pinList.contains(pin);

}

//Method to check the balance of an user account

public void checkBalance() throws SQLException {

String accountNumber = inputandvalidateAccountNo();

String accountPin = inputandvalidatePin();

String query = "SELECT account\_balance FROM Account WHERE account\_number = ? AND account\_pin = ?";

con = DatabaseConnection.getConnection();

pst = con.prepareStatement(query);

pst.setString(1,accountNumber);

pst.setString(2,accountPin);

rs = pst.executeQuery();

if(rs.next()) {

System.out.println("Account balance : " + rs.getFloat(1));

}

else {

System.err.println("Account not found");

}

}

//Method to create a new user account - new update

public void register() throws SQLException {

String accountNumber = inputandvalidateAccountNo();

// Check if the account number already exists

if (hasAccNoInDatabase(accountNumber)) {

System.err.println("An Account already exists with the account number. Please enter a new account number");

register(); // Re-register with a new account number

return; // Exit this call to prevent further execution

}

String accountPin = inputandvalidatePin();

double initialAmount = inputandvalidateAmount();

String mobileNumber = inputandvalidateMobileNo();

String query = "INSERT INTO Account(account\_number, account\_balance, account\_pin) VALUES(?,?,?)";

String getId = "SELECT account\_id FROM Account WHERE account\_number = ?";

con = DatabaseConnection.getConnection();

pst1 = con.prepareStatement(query);

pst1.setString(1, accountNumber);

pst1.setDouble(2, initialAmount);

pst1.setString(3, accountPin);

pst2 = con.prepareStatement(getId);

pst2.setString(1, accountNumber);

int rows = pst1.executeUpdate();

rs = pst2.executeQuery();

if (rows > 0 && rs.next()) {

System.out.println("Account has been created successfully with Account ID:" + rs.getInt(1));

System.out.println("OTP has been sent to the registered Mobile number. Kindly set your Username for the Account ID: " + rs.getInt(1));

} else {

System.err.println("Account cannot be created. Please check the input data are valid");

}

}

//Method to deposit money into an account

public void deposit() throws SQLException {

String accountNumber = inputandvalidateAccountNo();

String accountPin = inputandvalidatePin();

// Check if the account number already exists

if (!hasPinInAccount(accountPin,accountNumber)) {

System.err.println("The PIN entered is incorrect. Please enter your correct PIN");

deposit();

return;

}

double depositAmount = inputandvalidateAmount();

String query = "UPDATE Account SET account\_balance = account\_balance + ? WHERE account\_number = ? AND account\_pin = ?";

String getBalance = "SELECT account\_balance FROM Account WHERE account\_number = ?";

con = DatabaseConnection.getConnection();

pst = con.prepareStatement(query);

pst.setDouble(1,depositAmount);

pst.setString(2,accountNumber);

pst.setString(3,accountPin);

int rows = pst.executeUpdate();

pst = con.prepareStatement(getBalance);

pst.setString(1,accountNumber);

rs = pst.executeQuery();

if(rows > 0 && rs.next()) {

String recordTransactionQuery = "INSERT INTO Transaction (transaction\_date, account\_id, transaction\_type, amount) VALUES (NOW(), (SELECT account\_id FROM Account WHERE account\_number = ? AND account\_pin = ?), 'Deposit', ?)";

pst1 = con.prepareStatement(recordTransactionQuery);

pst1.setString(1, accountNumber);

pst1.setString(2, accountPin);

pst1.setDouble(3, depositAmount);

pst1.executeUpdate();

System.out.println("\nThe amount has been deposited");

System.out.println("Your current account balance is Rs." + rs.getDouble(1));

}

else {

System.err.println("Amount cannot be deposited. Please check the credentials and try again");

deposit();

}

}

//Method to withdraw money from an account

public void withdraw() throws SQLException {

String accountNumber = inputandvalidateAccountNo();

String accountPin = inputandvalidatePin();

double withdrawAmount = inputandvalidateAmount();

// Check if sufficient balance to withdraw

double currentBalance = getBalance(accountNumber);

if (currentBalance < (withdrawAmount + 100)) {

System.err.println("Cannot withdraw amount. Balance will become less than the minimum balance Rs.100");

System.out.println("Your current account balance is Rs." + currentBalance);

return;

}

// Update account balance

String updateQuery = "UPDATE Account SET account\_balance = account\_balance - ? WHERE account\_number = ? AND account\_pin = ?";

con = DatabaseConnection.getConnection();

pst = con.prepareStatement(updateQuery);

pst.setDouble(1, withdrawAmount);

pst.setString(2, accountNumber);

pst.setString(3, accountPin);

int rowsUpdated = pst.executeUpdate();

if (rowsUpdated > 0) {

// Record the transaction

String recordTransactionQuery = "INSERT INTO Transaction (transaction\_date, account\_id, transaction\_type, amount) VALUES (NOW(), (SELECT account\_id FROM Account WHERE account\_number = ? AND account\_pin = ?), 'Withdraw', ?)";

pst1 = con.prepareStatement(recordTransactionQuery);

pst1.setString(1, accountNumber);

pst1.setString(2, accountPin);

pst1.setDouble(3, withdrawAmount);

pst1.executeUpdate();

System.out.println("Please take the cash...");

System.out.println("Your current account balance is Rs." + (getBalance(accountNumber)));

} else {

System.err.println("Amount cannot be withdrawn.");

}

}

// Helper method to get account balance

private double getBalance(String accountNumber) throws SQLException {

String getBalanceQuery = "SELECT account\_balance FROM Account WHERE account\_number = ?";

con = DatabaseConnection.getConnection();

pst = con.prepareStatement(getBalanceQuery);

pst.setString(1, accountNumber);

ResultSet balanceResult = pst.executeQuery();

return balanceResult.next() ? balanceResult.getDouble(1) : 0;

}

}

**TRANSACTION STATEMENT CLASS:**

* This class will have the method to record each transaction when the user deposit or withdraws the amount from the account.

package com.atmproject;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Scanner;

public class TransactionStatement {

private static Connection con;

private static PreparedStatement pst;

private static ResultSet rs;

public void getCustomTransactionStatement() throws SQLException {

String accountNumber = AtmOperations.inputandvalidateAccountNo();

String accountPin = AtmOperations.inputandvalidatePin();

Scanner sc = new Scanner(System.in);

// Prompt the user for start date and time

System.out.print("Enter the start date (yyyy-MM-dd): ");

String startDateStr = sc.next();

System.out.print("Enter the start time (HH:mm:ss): ");

String startTimeStr = sc.next();

// Prompt the user for end date and time

System.out.print("Enter the end date (yyyy-MM-dd): ");

String endDateStr = sc.next();

System.out.print("Enter the end time (HH:mm:ss): ");

String endTimeStr = sc.next();

// Concatenate date and time for start and end

String formattedStartDate = startDateStr + " " + startTimeStr;

String formattedEndDate = endDateStr + " " + endTimeStr;

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

con = DatabaseConnection.getConnection();

try {

// Fetch account\_id based on account\_number

String accountIdQuery = "SELECT account\_id FROM Account WHERE account\_number = ?";

pst = con.prepareStatement(accountIdQuery);

pst.setString(1, accountNumber);

rs = pst.executeQuery();

int accountId = 0;

if (rs.next()) {

accountId = rs.getInt("account\_id");

} else {

System.err.println("Account not found.");

return;

}

// Convert start and end date/time to Date objects

Date startDate = dateFormat.parse(formattedStartDate);

Date endDate = dateFormat.parse(formattedEndDate);

// SQL to retrieve transactions between two dates for the given account\_id

String query = "SELECT \* FROM Transaction WHERE account\_id = ? AND transaction\_date BETWEEN ? AND ?";

pst = con.prepareStatement(query);

pst.setInt(1, accountId);

pst.setTimestamp(2, new java.sql.Timestamp(startDate.getTime()));

pst.setTimestamp(3, new java.sql.Timestamp(endDate.getTime()));

rs = pst.executeQuery();

System.out.println("Custom Transaction Statement for account number " + accountNumber + " between " + formattedStartDate + " and " + formattedEndDate + ":\n");

//System.out.format("%-15s%-20s%-15s%-10s%n", "Transaction ID", "Date", "Type", "Amount");

System.out.printf("+---------------+---------------------+---------------+------------+%n");

System.out.printf("|Transaction ID| Date | Type | Amount |%n");

System.out.printf("+---------------+---------------------+---------------+------------+%n");

while (rs.next()) {

int transactionId = rs.getInt("transaction\_id");

Date transactionDate = rs.getTimestamp("transaction\_date");

String transactionType = rs.getString("transaction\_type");

double amount = rs.getDouble("amount");

System.out.printf("| %-13d| %-20s| %-13s| %-10.2f|%n", transactionId, dateFormat.format(transactionDate), transactionType, amount);

}

System.out.printf("+---------------+---------------------+---------------+------------+%n");

} catch (Exception e) {

System.err.println("Error retrieving custom transaction statement: " + e.getMessage());

}

}

}

**MYSQL DATABASE RECORDS:**

There are two tables in the database created for the ATM Interface project. They are:

1. Account Table
2. Transaction Table

**Account Table:**

* The Account table stores the User’s bank account details such as account ID, account number, account balance, and account PIN.
* If a user creates an account the new account details will get inserted into the table.
* Similarly, if the user does any transaction, that will also be reflected in the account table.

Query to create account table:

CREATE TABLE Account (

account\_id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

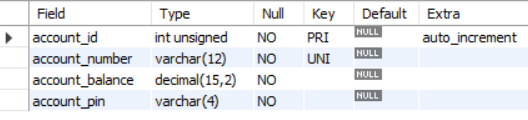
account\_number VARCHAR(20) UNIQUE NOT NULL,

account\_balance DECIMAL(15, 2),

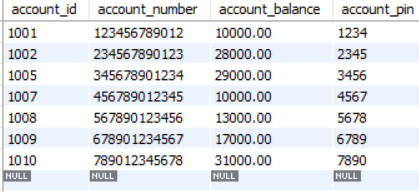
account\_pin VARCHAR(4) NOT NULL

) AUTO\_INCREMENT = 1000;

Account Table Description:



Account Table Records:



**Transaction Table:**

* The Transaction table stores the transactions made by the users with the details including the transaction id, transaction date, account id, transaction type (deposit or withdraw) and transaction amount.
* Whenever the user deposits or withdraw amount in his account the transaction table will be updated with the new record.

Query to create transaction table:

CREATE TABLE Transaction (

transaction\_id INT PRIMARY KEY AUTO\_INCREMENT,

transaction\_date TIMESTAMP,

account\_id INT UNSIGNED,

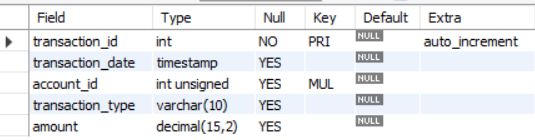
transaction\_type VARCHAR(10),

amount DECIMAL(15, 2),

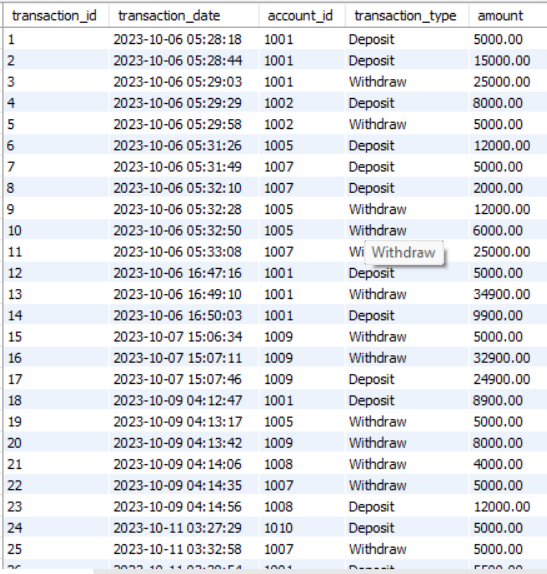
FOREIGN KEY (account\_id) REFERENCES Account(account\_id)

);

Transaction table Description:

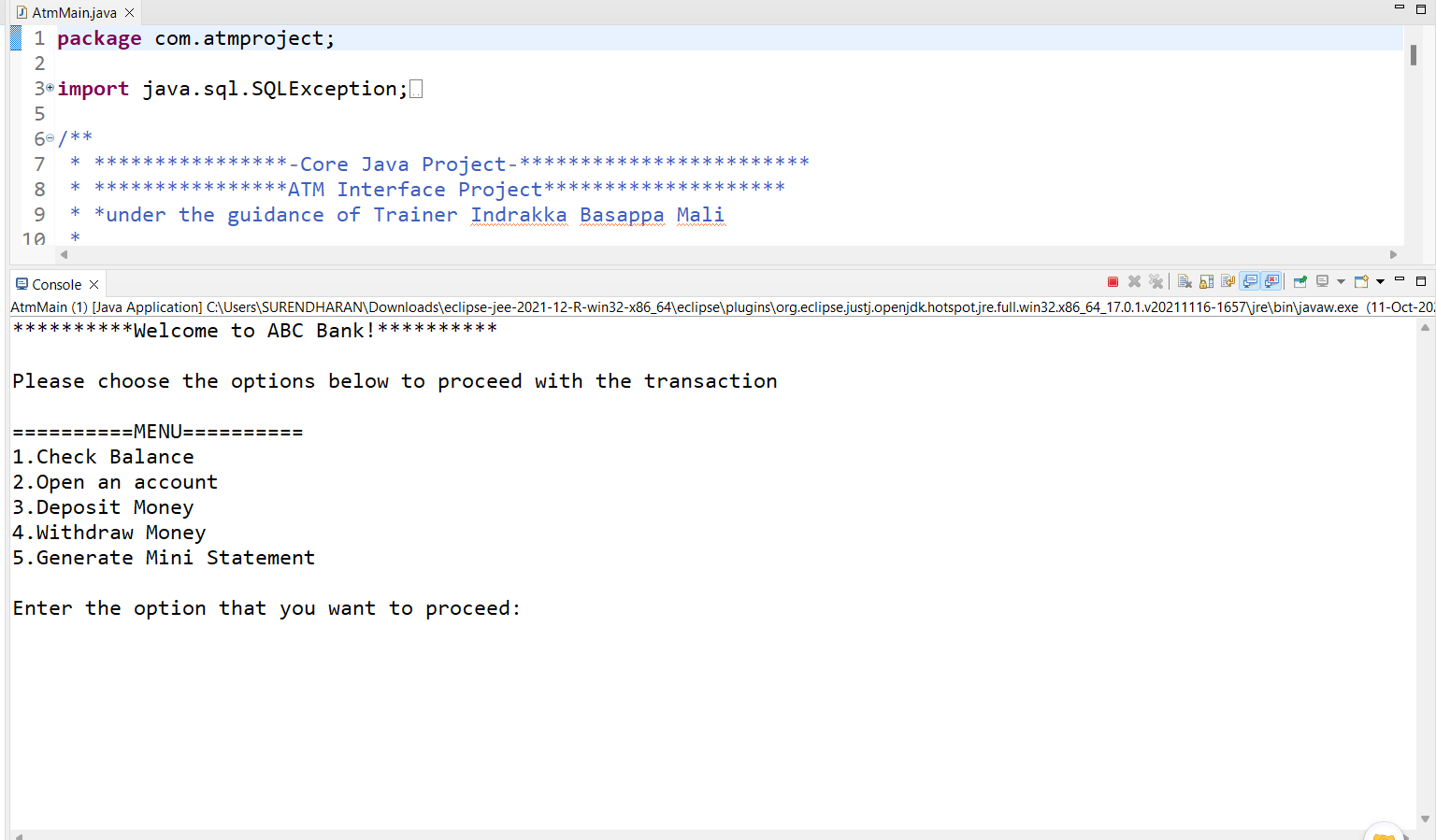


Transaction table Records:

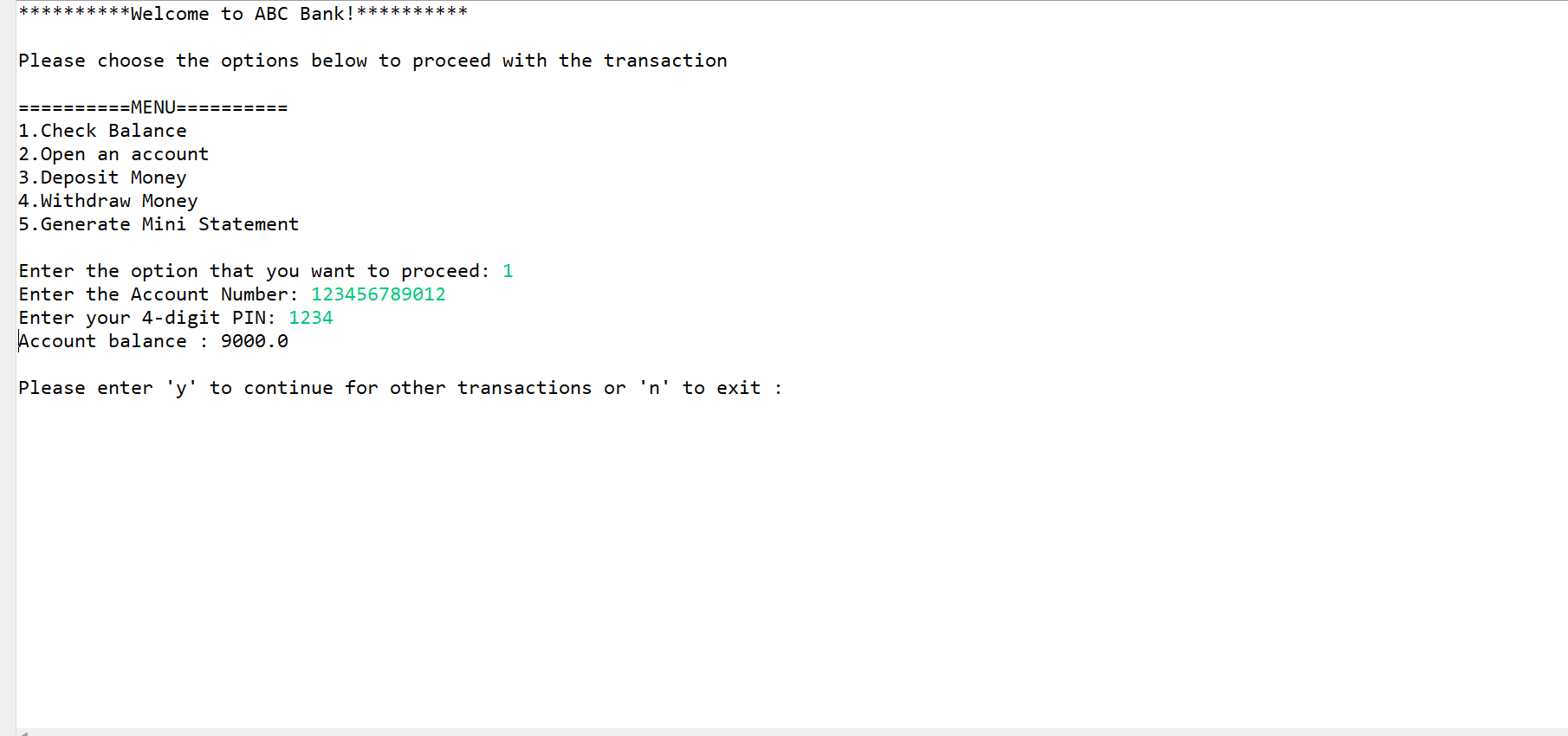


**RESULTS AND OUTPUT:**

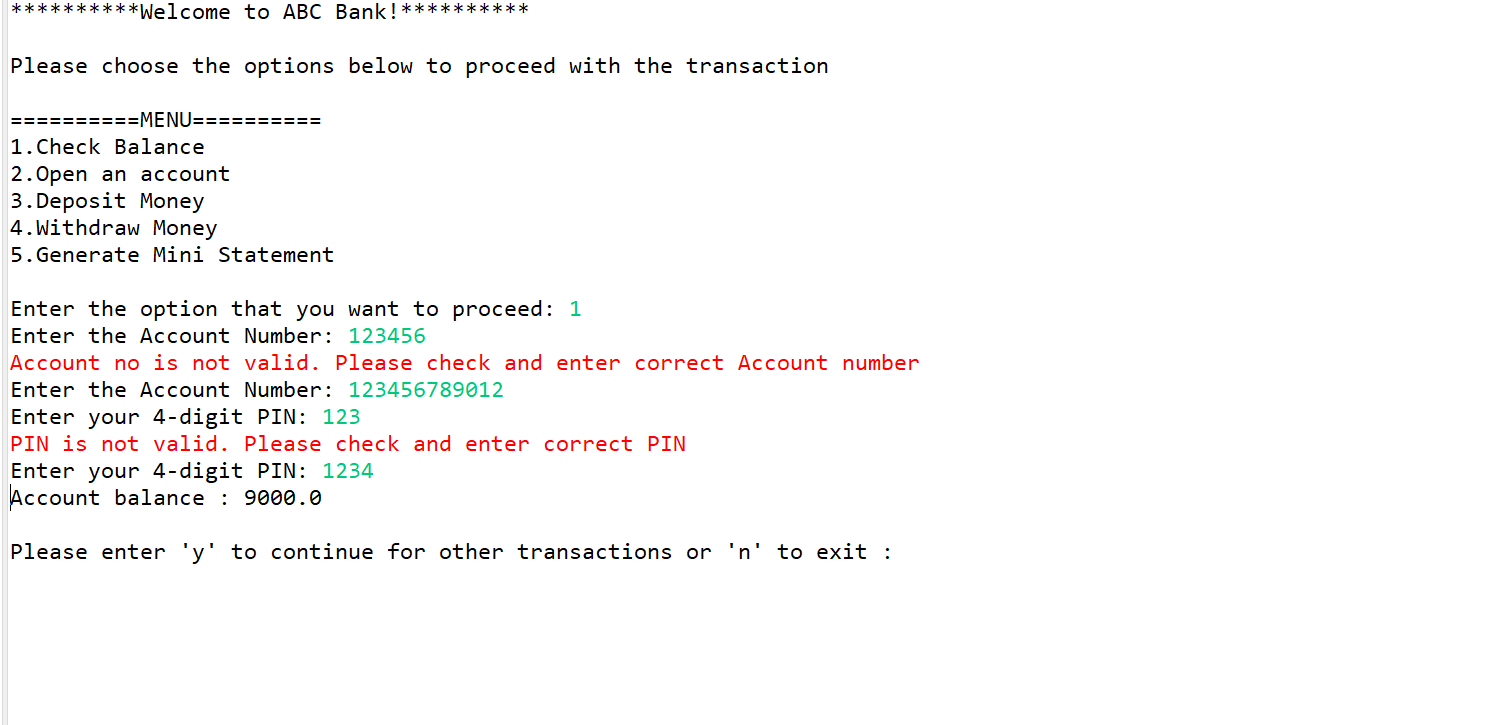
ATM MENU:



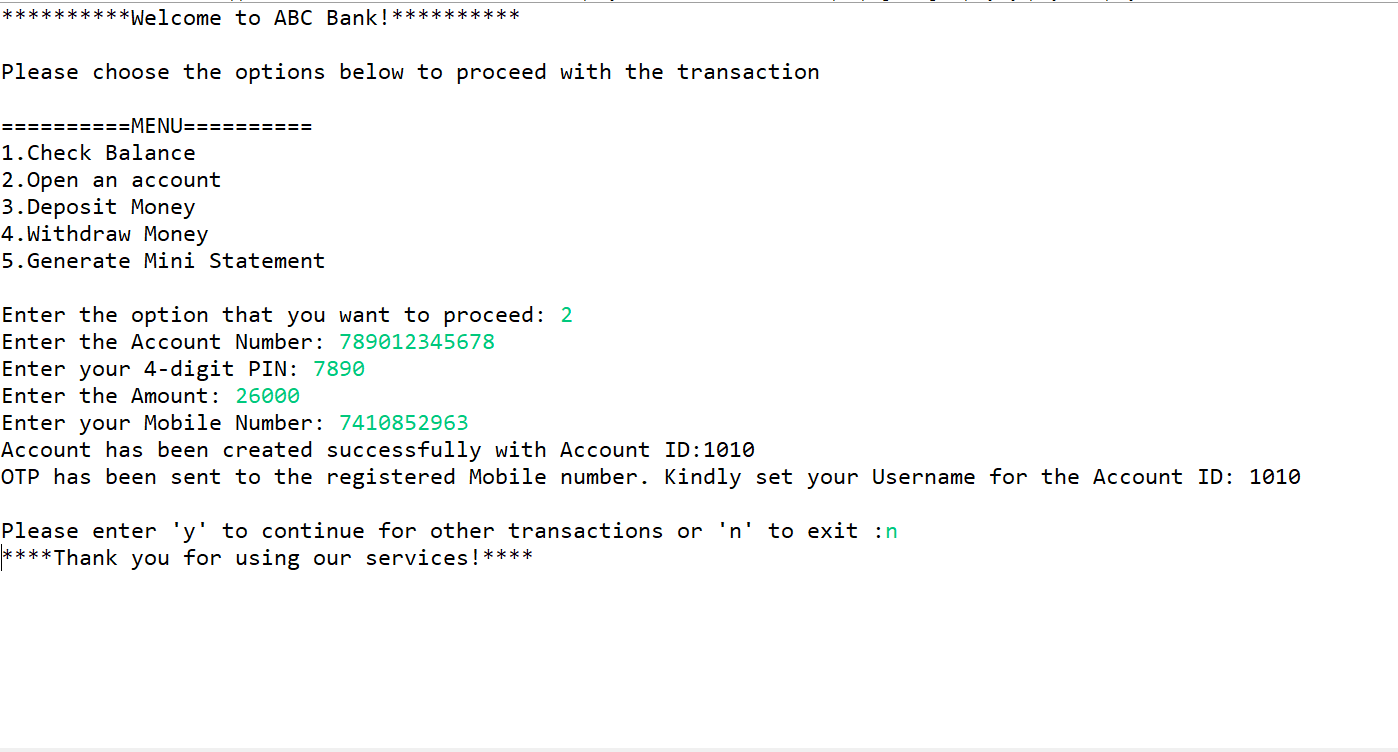
CHECK BALANCE IN ACCOUNT OPERATION:



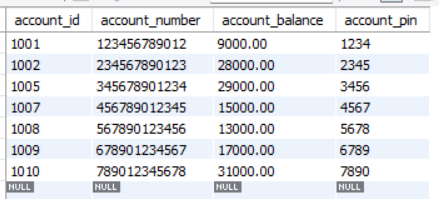
VALIDATION OF ACCOUNT NUMBER AND PIN:



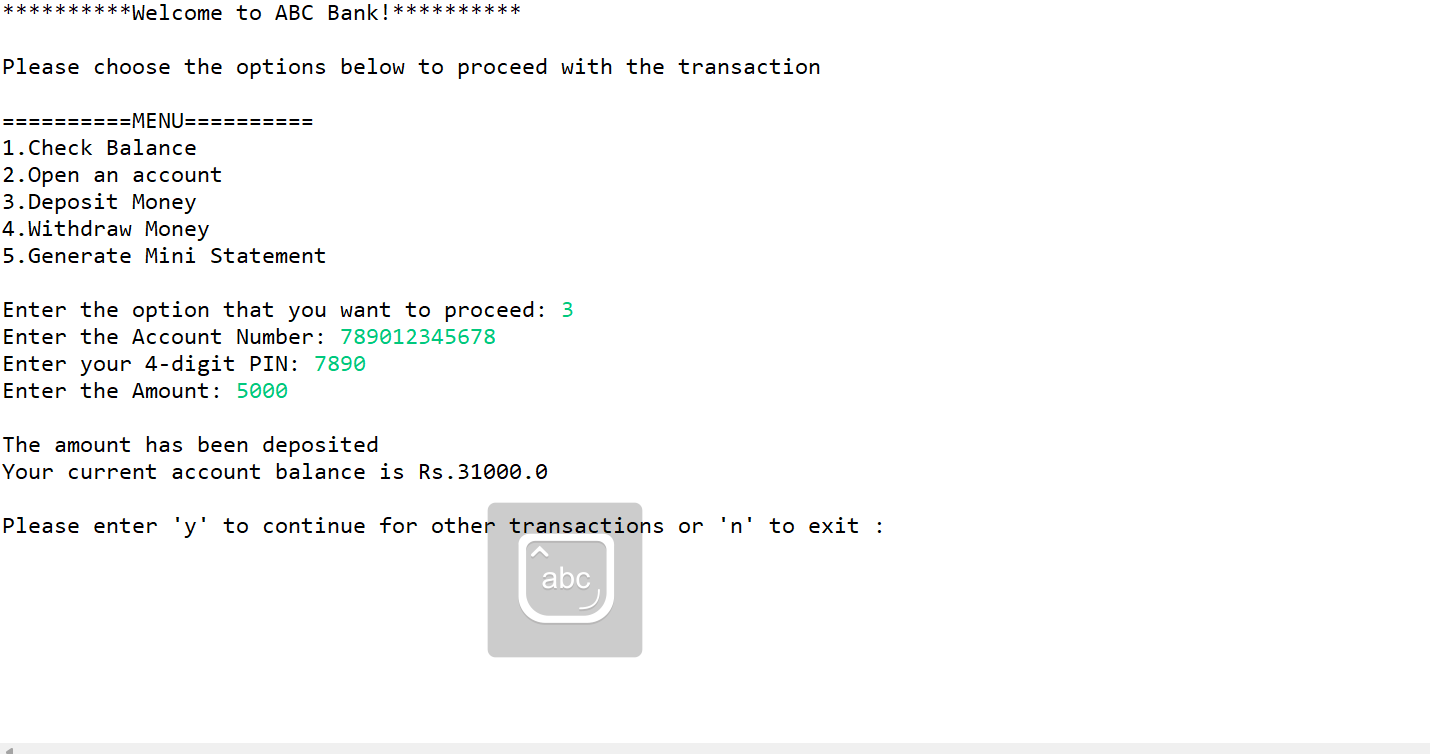
CREATE AN ACCOUNT OPERATION:



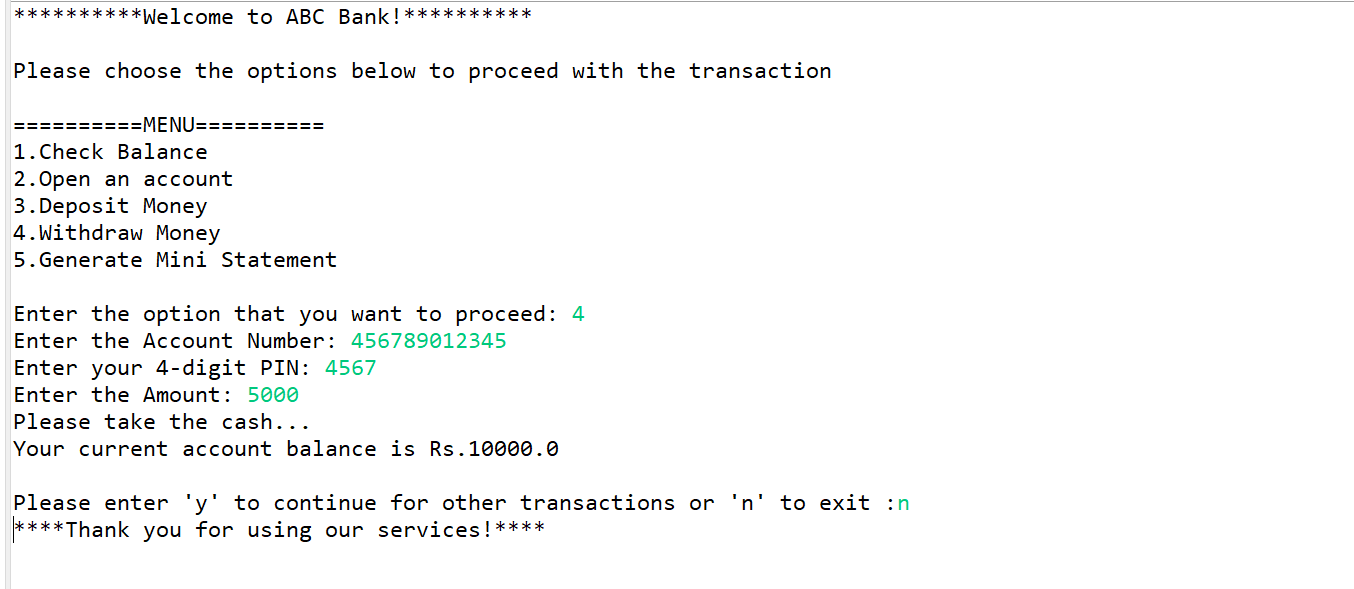
ACCOUNT TABLE AFTER CREATION OF ACCOUNT:



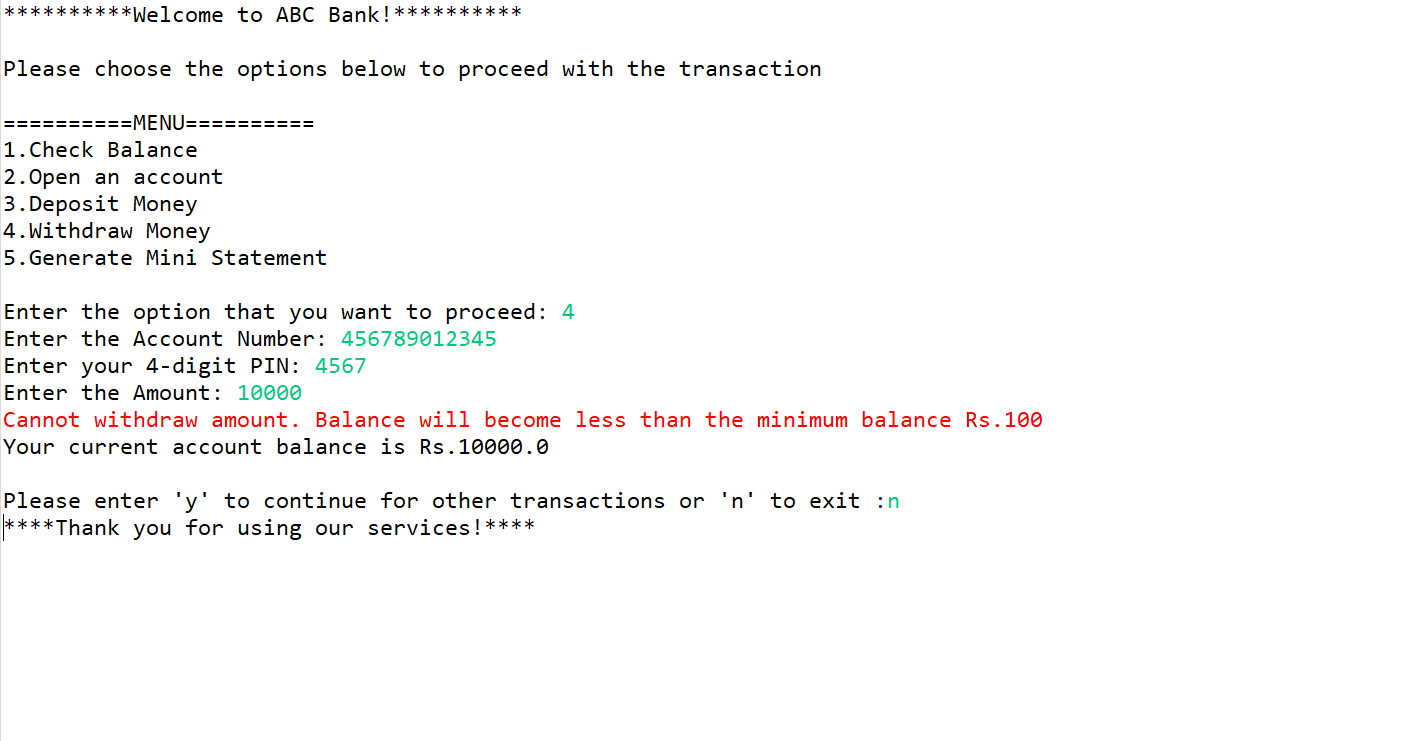
MONEY DEPOSIT OPERATION:



MONEY WITHDRAWAL OPERATION:



AMOUNT VALIDATION DURING WITHDRAWAL:



**CONCLUSION:**

* In conclusion, this ATM Interface Project demonstrates a robust and user-friendly system for conducting essential banking transactions.
* The system enables account registration, balance inquiries, fund deposits, withdrawals, and transaction history retrieval.
* Built using Java, it emphasizes security and validation at every step, enhancing the overall banking experience.
* The project underscores the significance of efficient software design, including database integration, user input handling, and intuitive interfaces, contributing to a reliable and functional ATM system.
* It provides a valuable foundation for further enhancements and serves as a practical application of core Java concepts.