JAVA PROJECT REPORT

(Project Term January-May 2023)

(ATM SIMULATION)

Submitted by

Sonu Kumar
Registration Number :12111009

Course Code- CSE310

Under the Guidance of

(Dr.A.Ranjith Kumar)

School of Computer Science and Engineering

DECLARATION

We hereby declare that the project work entitled ("*ATM SIMULATION*") is an authentic record of our own work carried out as requirements of Capstone Project for the award of B.Tech degree in CSE from Lovely Professional University, Phagwara, under the guidance of (Dr.A.Ranjith Kumar), during January to April 2023. All the information furnished in this capstone project report is based on our own intensive work and is genuine.

Name of Student 1: Sonu kumar
Registration Number: 12111009

Name of Student 2: Shanny Kumar Singh
Registration Number: 12213864

(Sonu kumar)
Date:20-04-2023

(Shanny Kumar Singh)
Date:20-04-2023

TABLE OF CONTENTS

ABSTRACT

An ATM (Automated Teller Machine) simulation is a computer program that replicates the functions and operations of an actual ATM machine. The simulation is designed to mimic the various processes that a customer undergoes when using an ATM, such as verifying the user's identity, displaying account balance, accepting deposits, and dispensing cash.

The purpose of an ATM simulation is to provide a safe, controlled environment for users to familiarize themselves with the functionality of an ATM without risking real money. The simulation can also be used to train new bank employees on how to use and troubleshoot the ATM.

An ATM simulation is typically created using programming languages such as Java or C++, and may utilize graphical user interfaces (GUI) to create a user-friendly experience. The simulation can be run on a personal computer, mobile device, or integrated into a banking system for customer use.

Overall, ATM simulations offer a valuable tool for banks and financial institutions to improve customer satisfaction and employee training, as well as to reduce errors and security risks associated with real-world ATM transactions.

1. INTRODUCTION

The aim of the ATM Simulation System project is to build a Java based ATM (Automated Teller Machine) Simulation System. The introduction of ATM's by various banks have brought about freedom from the interminable queues in front of withdrawal counters at banks. This ATM Simulation System requires the constant updating of records between the bank servers and a spread out network of ATM's.
Security is the foundation of a good ATM system. This system will provide for secure authenticated connections between users and the bank servers. The whole process will be automated right from PIN (Personal Identification Number) validation to transaction completion. ATM Simulation System will enable two important features of an ATM, reduction of human error in the banking system and the possibility of 24 hour personal banking. The card details and PIN database will be a secure module that will not be open to routine maintenance, the only possibility of access to this database will be through queries raised from an ATM in the presence of a valid bank ATM card.

Modern day banking means that a person who deposits money in a bank branch may actually require to withdraw the same amount some 100 – 200 km's away within 6 hours of the deposit. This requires the presence of accurate and constantly updated bank records. This will be possible due to a secure internet connection from the different bank branches to a central bank server connected further in a secure fashion to different ATM's at different locations. This system will have a Graphical User Interface which will make the whole process user friendly. There will be a small learning curve for new users but tutorials available at the ATM will ensure anyone can be an ATM expert in 2 minutes.

Any computerized system is susceptible to failure but failure in an ATM system can be catastrophic and even lead to monetary loss. Scheduled on – site maintenance can ensure secure software upgrades and help avoid potential failure. The ATM will be programmed to notify the bank about impending maintenance and will go into a lock down mode in the event of delays of more than 7 days in the maintenance.

2. Survey of Existing System

The existing ATM systems can be categorized into two types: stand-alone ATM systems and networked ATM systems.

Stand-alone ATM systems are machines that are not connected to any central banking network. They are typically found in retail establishments or other locations where a bank branch is not available. These ATMs can perform basic transactions such as cash withdrawals, balance inquiries, and deposits. However, they may not offer more advanced features such as bill payments or money transfers.

Networked ATM systems, on the other hand, are connected to a central banking network and can offer a wider range of services. These ATMs can be found in bank branches, shopping centers, and other high-traffic locations. They can perform all the basic transactions as well as more advanced features such as bill payments, money transfers, and check cashing.

Most modern ATM systems use a combination of hardware and software to provide secure and reliable service. The hardware includes the ATM machine itself, a card reader, a keypad, a cash dispenser, and a receipt printer. The software includes the operating system, application software, and security software.

Security is a critical aspect of ATM systems, and many existing systems employ multiple layers of security to protect against fraud and theft. This can include measures such as PIN authentication, biometric identification, and encryption of sensitive data.

Overall, ATM systems have become an essential part of the banking industry, providing convenient and secure access to financial services for millions of people around the world. As technology continues to evolve, we can expect to see further advances in ATM systems, including new features and improved security measures..

3. TECHNOLOGY

JFrame:

JFrame is a class in the Java Swing library that provides a basic window for creating graphicaluser interfaces (GUI) in Java. It is a top-level container that can be used to hold other Swing components such as buttons, labels, text fields, and other GUI elements.

JFrame provides a platform-independent way to create GUI applications that can run on differentoperating systems. It has a wide range of functionality such as displaying images, menus, and toolbars. JFrame can also be customized with different colors, fonts, and sizes to create visually appealing user interfaces.

JFrame inherits from the java.awt. Frame class, which is part of the Abstract Window
Toolkit (AWT) in Java. However, JFrame is preferred over Frame because it is built.
on top of Swing, which provides a richer set of components and is more flexible and
customizable than AWT.

To create a JFrame in Java, developers must first import the necessary classes from the Swing library. Then, they can create a new JFrame object and add other components to it as needed. Developers can also set properties of the JFrame such as its size, title, and default close operation.

Overall, JFrame is a versatile class in the Java Swing library that provides a basic window for creating GUI applications in Java. Its flexibility and customizability make it an essential tool for developers who want to create visually appealing and use**r** friendly application.

4.FUNCTIONALITY:

In this project users have the following functionality: -

1. User has the option to use the saving account or current account based on his/her choice.

2. User can withdraw money by entering the correct amount.

3. Users can deposit money in both account types.

4. User can check the left balance from both accounts after withdrawing.

5. User can choose any choice based on his/her requirements.

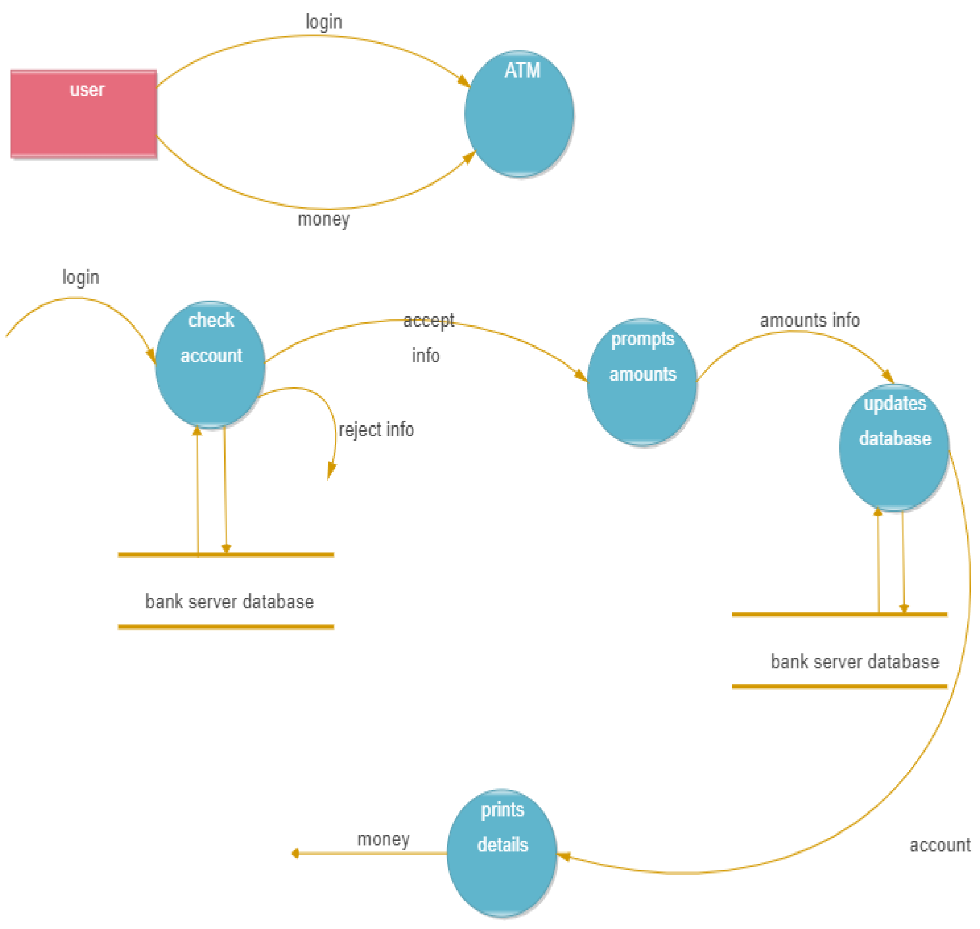
6. Provide or implement the idea of Java inheritance which can help others to learn Javainheritance topics.

- ATM users can deposit money to the bank by choosing the deposit option. The java program willsimply get the deposit amount from the user and add the money to the user's account.

- Users can withdraw money from their bank account through the program by selecting the withdrawal option. After a successful transaction, the ATM machine will deduct the amount fromthe central bank account.

- The user can also check their existing bank account total balance using the checking accountbalance option.

5.Flowchart for ATM



Data Flow Diagram for An ATM System

5.MODULES

1.Customer

* Deposit Cash – To deposit cash

* Transfer Cash – Transfer cash from user account to another account

* Check Balance – Check balacce after cash withdrawl or deposit

* Withdraw Cash – To withdraw cash

* Fast Cash – To get fast cash

* Exit – To close customer pannel

2.Admin

* Create Account – To create account

* Delete Account – Delete existing account

* Update Account – Update account after transactions

* Search Account – To find whether the account exit or not

* View Report – To know details about user

* Exit – To close the admin pannel

6. Results:

6.1 Main window

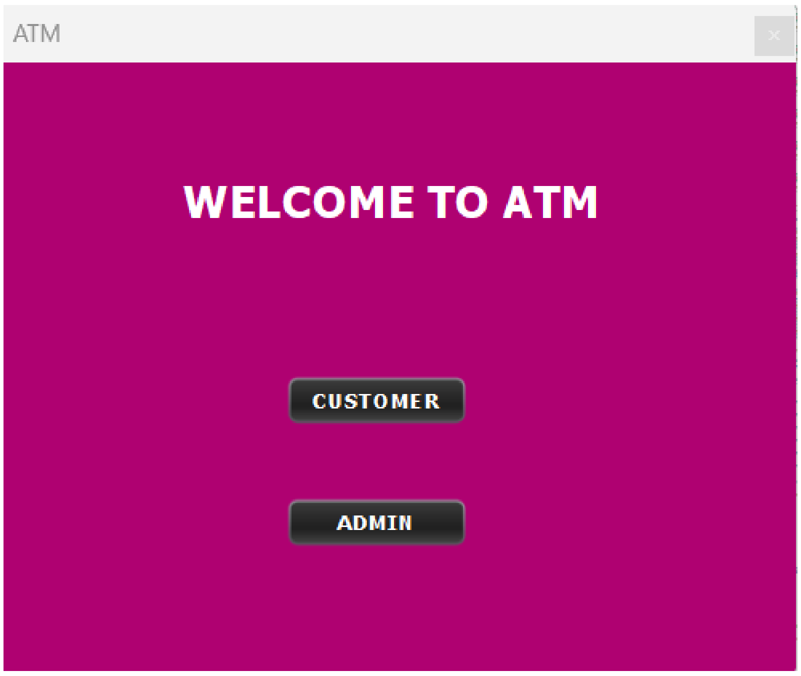


Figure -1

6.2 Customer window

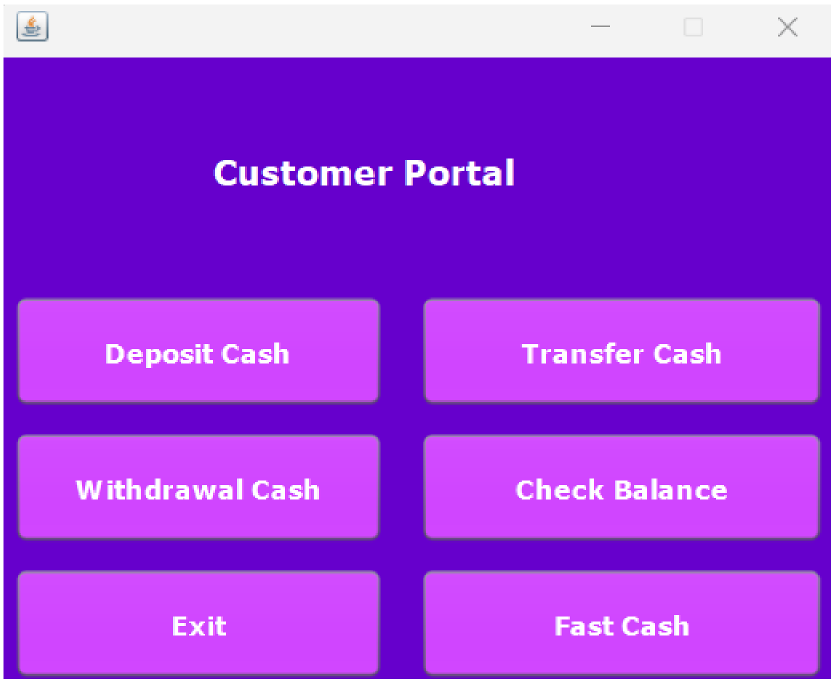


Figure -2

6.3 Admin window

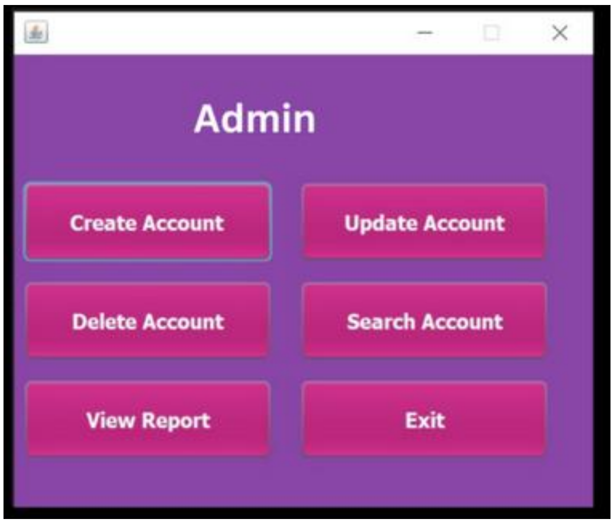
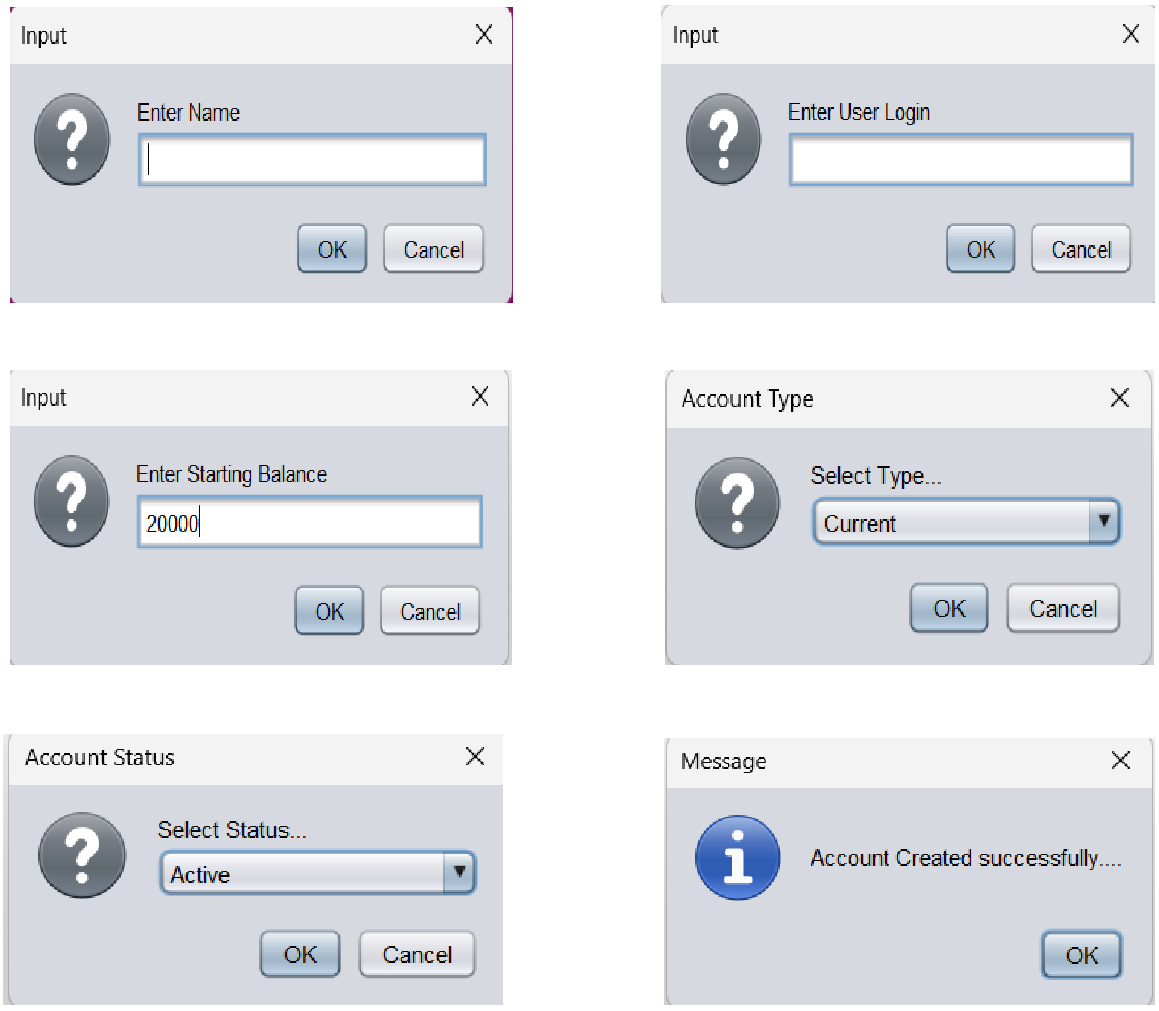


Figure -3

6.4 Create account



Enter Name

Enter User Login

Enter Starting Balance

20000

Select Type...

Current

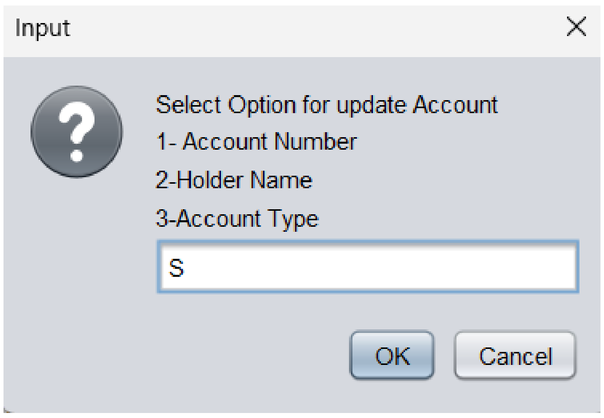
Select Status...

Active

Account Created successfully....

Figure -4

6.5 Update account

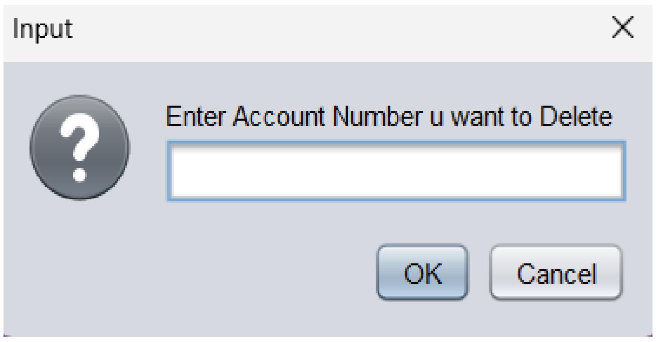

Input

Select Option for update Account
1- Account Number
2-Holder Name
3-Account Type

S

Figure -5

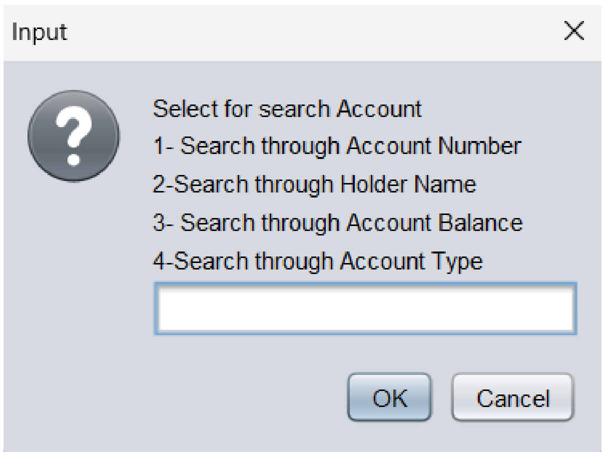
6.6 Delete account


Input

Enter Account Number u want to Delete

Figure -6

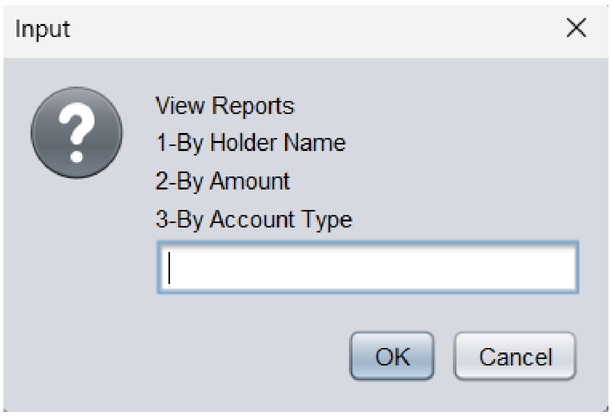
6.7 Searching account


Input

Select for search Account
1- Search through Account Number
2-Search through Holder Name
3- Search through Account Balance
4-Search through Account Type

Figure -7

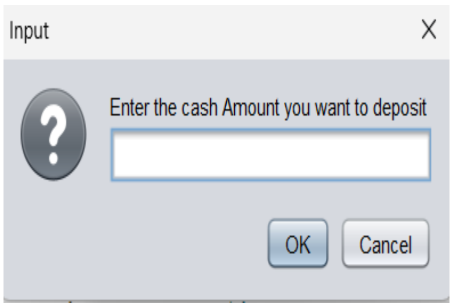
6.8 View report



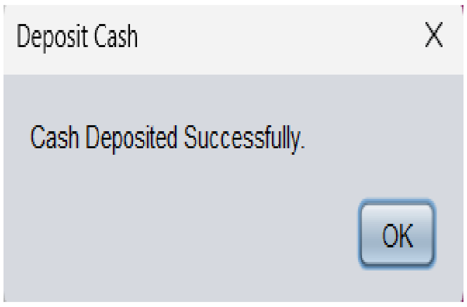
View Reports
1-By Holder Name
2-By Amount
3-By Account Type

Figure -8

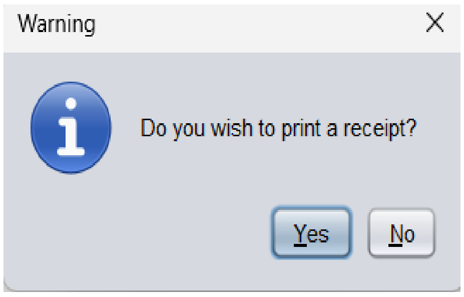
6.9 Deposit cash



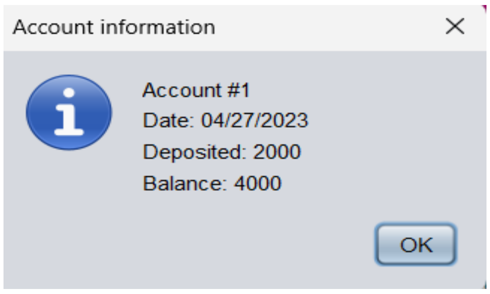
Enter the cash Amount you want to deposit



Cash Deposited Successfully.



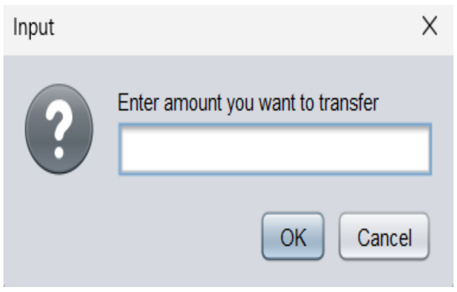
Do you wish to print a receipt?



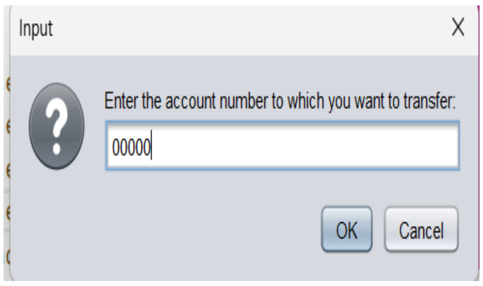
Account #1
Date: 04/27/2023
Deposited: 2000
Balance: 4000

Figure -9

6.10 Transfer cash



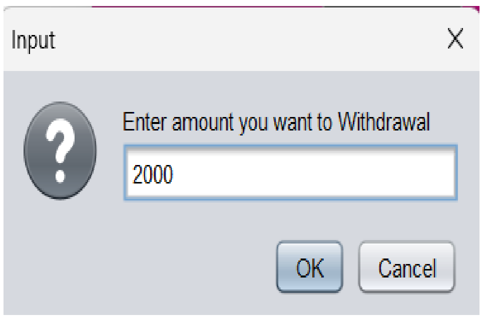
Enter amount you want to transfer



Enter the account number to which you want to transfer:
00000

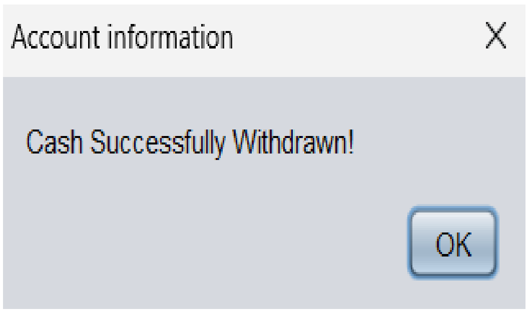
Figure -10

6.11 Withdrawal cash

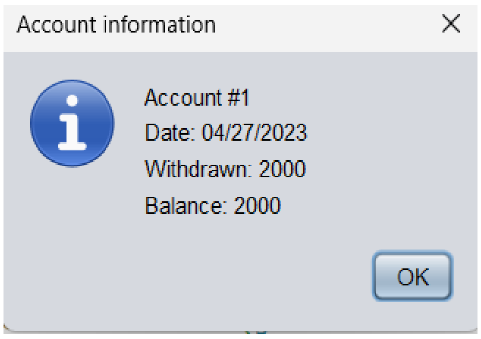

Input

Enter amount you want to Withdrawal

2000


Account information

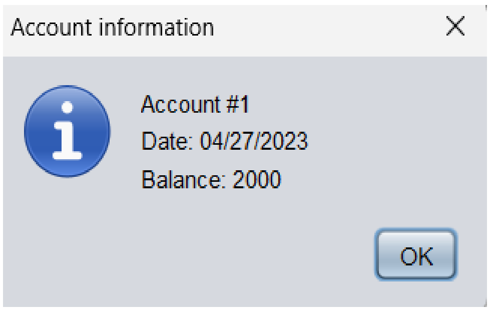
Cash Successfully Withdrawn!


Account information

Account #1
Date: 04/27/2023
Withdrawn: 2000
Balance: 2000

Figure -11

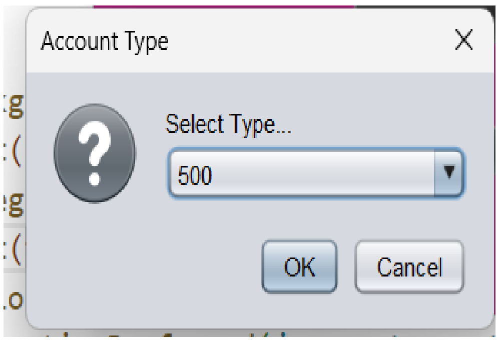
6.12 Check balance


Account information

Account #1
Date: 04/27/2023
Balance: 2000

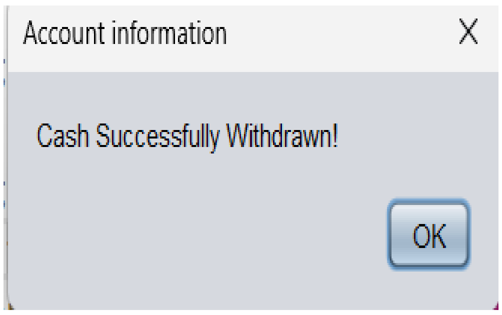
Figure -12

6.13 Fast cash

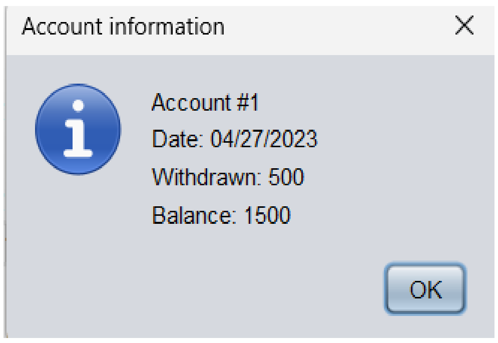

Account Type

Select Type...

500


Account information

Cash Successfully Withdrawn!


Account information

Account #1
Date: 04/27/2023
Withdrawn: 500
Balance: 1500

Figure -13

7. Conclusion

In conclusion, the ATM simulation is an excellent tool for banks and financial institutions to train their employees and customers on the use of ATM machines. The simulation provides a realistic and safe environment for users to practice various ATM operations such as deposits, withdrawals, and balance inquiries.

The simulation also helps users to understand the security features of an ATM machine, such as the use of PINs and card verification codes, to ensure the safety of their transactions. Moreover, the simulation can be customized to fit the specific needs of the institution and to include new features as they become available.

Overall, the ATM simulation is a valuable tool that helps to improve the efficiency, security, and convenience of ATM transactions for both the financial institution and its customers.

8. Reference

- www.google.co.in
- https://www.javatpoint.com
- https://chat.openai.com/chat/b8a39b1b-6f36-4678-9e98-b6fb7310d346

Code:

```java
import java.util.*;
import java.text.SimpleDateFormat;
import javax.swing.*;


class CUSTOMER extends JFrame {

    ATM var;

    public CUSTOMER(ATM a) {
        var = a;
        this.user = new ArrayList<Customer_Data>();
        initComponents();
    }
    List<Customer_Data> user;

    boolean onlyContainsNumbers(String text) {
        try {
            Long.parseLong(text);
            return true;
        } catch (NumberFormatException ex) {
            return false;
        }
    }

    // fast cash method
    public void fastCash() {
        int ab;
        String[] s10 = {"500", "1000", "2000", "5000", "10000", "15000",
"20000", ""};
        s10[7] = (String) JOptionPane.showInputDialog(null, "Select Type...",
"Account Type", JOptionPane.QUESTION_MESSAGE, null, s10, s10[0]);

        switch (s10[7]) {
            // if user select 500, we check if the user have enough balance to
withdraw 500
            case "500":
                if (var.var1.startingBalance > 500 && var.var1.startingBalance
> 0) {
                    var.var1.startingBalance = var.var1.startingBalance - 500;
                    JOptionPane.showMessageDialog(null, "Cash Successfully
Withdrawn! ", "Account information", JOptionPane.PLAIN_MESSAGE);

                    ab = JOptionPane.showConfirmDialog(null, "Do you wish to
print a receipt?", "Warning", JOptionPane.YES_NO_OPTION,
JOptionPane.INFORMATION_MESSAGE);
```

```
                    if (ab == 0) {
                        JOptionPane.showMessageDialog(null, "Account #" +
var.var1.accountNum + "\nDate: " + new
SimpleDateFormat("MM/dd/yyyy").format(Calendar.getInstance().getTime()) +
"\nWithdrawn: " + "500" + "\nBalance: " + var.var1.startingBalance, "Account
information", JOptionPane.INFORMATION_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "You do not have
enough Amount\nCurrent Amount : " + var.var1.startingBalance, "Account
information", JOptionPane.ERROR_MESSAGE);
                }
                break;
            // if user select 1000, we check if the user have enough balance
to withdraw 1000
            case "1000":
                if (var.var1.startingBalance > 1000 &&
var.var1.startingBalance > 0) {
                    var.var1.startingBalance = var.var1.startingBalance -
1000;
                    JOptionPane.showMessageDialog(null, "Cash Successfully
Withdrawn! ", "Account information", JOptionPane.PLAIN_MESSAGE);

                    ab = JOptionPane.showConfirmDialog(null, "Do you wish to
print a receipt?", "Warning", JOptionPane.YES_NO_OPTION,
JOptionPane.INFORMATION_MESSAGE);
                    if (ab == 0) {
                        JOptionPane.showMessageDialog(null, "Account #" +
var.var1.accountNum + "\nDate: " + new
SimpleDateFormat("MM/dd/yyyy").format(Calendar.getInstance().getTime()) +
"\nWithdrawn: " + "1000" + "\nBalance: " + var.var1.startingBalance, "Account
information", JOptionPane.INFORMATION_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "You do not have
enough Amount\nCurrent Amount : " + var.var1.startingBalance, "Account
information", JOptionPane.ERROR_MESSAGE);
                }
                break;
            // if user select 2000, we check if the user have enough balance
to withdraw 2000, and so on
            case "2000":
                if (var.var1.startingBalance > 2000 &&
var.var1.startingBalance > 0) {
                    var.var1.startingBalance = var.var1.startingBalance -
2000;
                    JOptionPane.showMessageDialog(null, "Cash Successfully
Withdrawn! ", "Account information", JOptionPane.PLAIN_MESSAGE);
```

```
                    ab = JOptionPane.showConfirmDialog(null, "Do you wish to 
print a receipt?", "Warning", JOptionPane.YES_NO_OPTION, 
JOptionPane.INFORMATION_MESSAGE);
                    if (ab == 0) {
                        JOptionPane.showMessageDialog(null, "Account #" + 
var.var1.accountNum + "\nDate: " + new 
SimpleDateFormat("MM/dd/yyyy").format(Calendar.getInstance().getTime()) + 
"\nWithdrawn: " + "2000" + "\nBalance: " + var.var1.startingBalance, "Account 
information", JOptionPane.INFORMATION_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "You do not have 
enough Amount\nCurrent Amount : " + var.var1.startingBalance, "Account 
information", JOptionPane.ERROR_MESSAGE);
                }
                break;
            case "5000":
                if (var.var1.startingBalance > 5000 && 
var.var1.startingBalance > 0) {
                    var.var1.startingBalance = var.var1.startingBalance - 
5000;
                    JOptionPane.showMessageDialog(null, "Cash Successfully 
Withdrawn! ", "Account information", JOptionPane.PLAIN_MESSAGE);

                    ab = JOptionPane.showConfirmDialog(null, "Do you wish to 
print a receipt?", "Warning", JOptionPane.YES_NO_OPTION, 
JOptionPane.INFORMATION_MESSAGE);
                    if (ab == 0) {
                        JOptionPane.showMessageDialog(null, "Account #" + 
var.var1.accountNum + "\nDate: " + new 
SimpleDateFormat("MM/dd/yyyy").format(Calendar.getInstance().getTime()) + 
"\nWithdrawn: " + "5000" + "\nBalance: " + var.var1.startingBalance, "Account 
information", JOptionPane.INFORMATION_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "You do not have 
enough Amount\nCurrent Amount : " + var.var1.startingBalance, "Account 
information", JOptionPane.ERROR_MESSAGE);
                }
                break;
            case "10000":
                if (var.var1.startingBalance > 10000 && 
var.var1.startingBalance > 0) {
                    var.var1.startingBalance = var.var1.startingBalance - 
10000;
                    JOptionPane.showMessageDialog(null, "Cash Successfully 
Withdrawn! ", "Account information", JOptionPane.PLAIN_MESSAGE);
```

```
                    ab = JOptionPane.showConfirmDialog(null, "Do you wish to
print a receipt?", "Warning", JOptionPane.YES_NO_OPTION,
JOptionPane.INFORMATION_MESSAGE);

                    if (ab == 0) {
                        JOptionPane.showMessageDialog(null, "Account #" +
var.var1.accountNum + "\nDate: " + new
SimpleDateFormat("MM/dd/yyyy").format(Calendar.getInstance().getTime()) +
"\nWithdrawn: " + "10000" + "\nBalance: " + var.var1.startingBalance, "Account
information", JOptionPane.INFORMATION_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "You do not have
enough Amount\nCurrent Amount : " + var.var1.startingBalance, "Account
information", JOptionPane.ERROR_MESSAGE);
                }
                break;
            case "15000":
                if (var.var1.startingBalance > 15000 &&
var.var1.startingBalance > 0) {
                    var.var1.startingBalance = var.var1.startingBalance -
15000;
                    JOptionPane.showMessageDialog(null, "Cash Successfully
Withdrawn! ", "Account information", JOptionPane.PLAIN_MESSAGE);

                    ab = JOptionPane.showConfirmDialog(null, "Do you wish to
print a receipt?", "Warning", JOptionPane.YES_NO_OPTION,
JOptionPane.INFORMATION_MESSAGE);

                    if (ab == 0) {
                        JOptionPane.showMessageDialog(null, "Account #" +
var.var1.accountNum + "\nDate: " + new
SimpleDateFormat("MM/dd/yyyy").format(Calendar.getInstance().getTime()) +
"\nWithdrawn: " + "15000" + "\nBalance: " + var.var1.startingBalance, "Account
information", JOptionPane.INFORMATION_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "You do not have
enough Amount\nCurrent Amount : " + var.var1.startingBalance, "Account
information", JOptionPane.ERROR_MESSAGE);
                }
                break;
            case "20000":
                if (var.var1.startingBalance > 20000 &&
var.var1.startingBalance > 0) {
                    var.var1.startingBalance = var.var1.startingBalance -
20000;
```

```
                    JOptionPane.showMessageDialog(null, "Cash Successfully
Withdrawn! ", "Account information", JOptionPane.PLAIN_MESSAGE);

                    ab = JOptionPane.showConfirmDialog(null, "Do you wish to
print a receipt?", "Warning", JOptionPane.YES_NO_OPTION,
JOptionPane.INFORMATION_MESSAGE);

                    if (ab == 0) {
                        JOptionPane.showMessageDialog(null, "Account #" +
var.var1.accountNum + "\nDate: " + new
SimpleDateFormat("MM/dd/yyyy").format(Calendar.getInstance().getTime()) +
"\nWithdrawn: " + "20000" + "\nBalance: " + var.var1.startingBalance, "Account
information", JOptionPane.INFORMATION_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "You do not have
enough Amount\nCurrent Amount : " + var.var1.startingBalance, "Account
information", JOptionPane.ERROR_MESSAGE);
                }
                break;
        }
    }
```