Create a fully functional ATM interface using Java

Introduction

Automated Teller Machines (ATMs) have become an integral part of modern banking systems, providing customers with quick and efficient access to banking services. An ATM interface is the user-facing component that allows individuals to perform financial transactions such as cash withdrawals, deposits, balance inquiries, and fund transfers. A fully functional ATM interface must be intuitive and responsive, allowing users to interact with the system easily. The interface typically consists of a graphical or text-based display, a numeric keypad, and various function buttons. Users interact with the ATM by inserting their debit or credit card, entering their Personal Identification Number (PIN), and selecting their desired transactions.

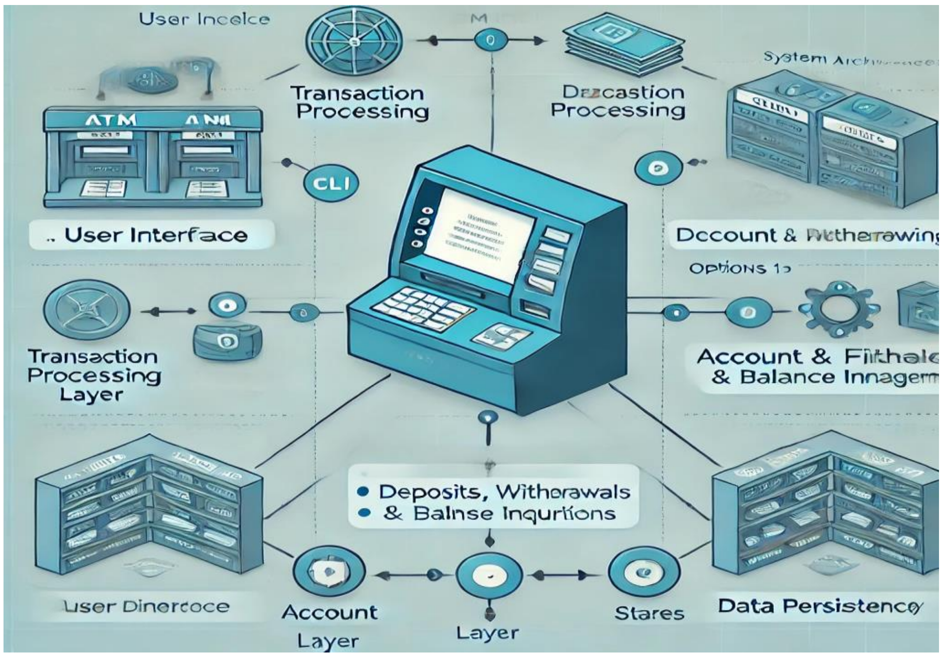
ATM interface is ability to handle different types of transactions. Common operations include:

1. **Deposit Money –** Adding funds to a bank account by inserting cash or transferring electronically through the ATM system securely.
2. **Withdraw Money** – Removing a specified amount from the account, ensuring sufficient balance and security measures before dispensing cash.
3. **Transaction History** – A record of past financial activities, including deposits, withdrawals, and transfers, for user reference and security tracking.
4. **Check Balance** – Viewing the current available funds in an account to manage expenses and ensure sufficient money for future transactions.
5. **Exit** – Securely logging out of the ATM system after completing transactions to prevent unauthorized access and maintain privacy.

Modules

1. **Account Management Module**
    - **Create Account**: Users can create an account with a name, username, PIN, and initial balance.
2. **Transaction Processing Module**
    - **Deposit Money**: Adds money to the account balance and records the transaction.
    - **Withdraw Money**: Users should be able to withdraw money from their account and the system should ensure that withdrawal amounts are in multiples of 100.
3. **Transaction History Module**
    - **View Past Transactions**: Reads and displays transaction records.
4. **Balance Inquiry Module**
    - **Check Current Balance**: Displays the user's available balance.
5. **System Exit Module**
    - **Exit ATM System**: Provides a way to safely terminate the program.

System Architecture:



1. User Interface

- The user interacts with the system through the console (System.out.println() statements).
- The interface provides options:
    1. Create an account (createAcc() method).
    2. Deposit money (deposit() method).
    3. Withdraw money (withdraw() method).
    4. View transaction history (transactionHistory() method).
    5. Check balance (checkBalance() method).
    6. Exit.

2. Transaction Processing Layer

- The ATM class handles transactions.
- When a user selects an option, it routes the request to the appropriate method
    1. Deposit (deposit() method adds funds).
    2. Withdraw (withdraw() method checks balance and deducts funds).

3.Deposit & Withdrawal Processing

- The deposit() method updates the balance.
- The withdraw() method checks available funds and validates withdrawal conditions (like multiples of 100).

4.Account & Balance Management

- The checkBalance() method displays the user's current balance.
- The transactionHistory() method retrieves past transactions from a file.

5.Data Persistence Layer

- Transactions and account details are stored in a text file (FileWriter and BufferedWriter methods).
- Each transaction (deposit/withdrawal) is appended to the account's respective file.

6.Account Layer

- The Account class creates and maintains user details like:
    1. Account number.
    2. Holder name.

3. PIN.
4. Balance.

7. System Flow (Complete ATM Operation)

1. **User Interaction** → Selects an option from the menu.
2. **Transaction Processing** → ATM processes deposit/withdrawal.
3. **Deposit & Withdrawal Processing** → Ensures the request is valid.
4. **Account & Balance Management** → Updates account details.
5. **Data Persistence** → Stores transaction records for future reference.

Advantages

- ATM services are available 24×7.
- ATMs offer global access, enabling you to carry out numerous financial activities easily, regardless of where you are. This makes banking easy and accessible, improving customer experience.
- Using ATMs encourages you to manage your transactions independently and be in control of your finances.
- Another advantage of ATM machines is faster and more convenient financial transactions through simple and rapid operations including cash withdrawals, balance checks, and more.

Disadvantages

- **Technical Issues:** ATMs may face technical issues like malfunctions and network problems, resulting in service disruptions. These may inconvenience you if you're requesting rapid cash access.
- **Limited Cash Availability:** ATMs have daily withdrawal limitations and may run out of cash during periods of high usage, limiting your access to your funds. This can be especially difficult in emergencies where you may need larger amounts.

Source code

```
import java.io.*;

import java.util.*;

class Account {

    static int acc_number = 1111;

    String acc_holder_name;

    int pin;

    double balance;

    String unique_id;

    int a_no;

   void createAcc() {

    a_no = acc_number;

    Scanner in = new Scanner(System.in);

    System.out.println("Enter account holder name:");

     acc_holder_name = in.nextLine();

     System.out.println("Enter Username:");

     unique_id = in.nextLine();

       do {

       System.out.println("Enter a 4-digit PIN:");

        pin = in.nextInt();

        } while (String.valueOf(pin).length() != 4);

     System.out.print("Enter initial deposit amount: ");

      balance = in.nextDouble();

      System.out.println("\nCongratulations! Account Successfully Created.\n");

      System.out.println("Account Details:\nAccount Number: " + a_no + "\nAccount Holder
Name: " + acc_holder_name + "\nBalance: " + balance);

   String fileName = a_no + ".txt";

     try (FileWriter writer = new FileWriter(fileName)) {

            writer.write("Account Created\n");

            writer.write("Account Number: " + a_no + "\n");
            writer.write("USER ID: " + unique_id + "\n");
            writer.write("Account Holder Name: " + acc_holder_name + "\n");
            writer.write("PIN: " + pin + "\n");
            writer.write("Balance: " + balance + "\n");
            writer.write("Date: " + new Date() + "\n\n");
        } catch (IOException e) {
            System.out.println("Error creating file: " + fileName);
        }
        acc_number++;
    }
}

class ATM {
    void deposit(Account acc, double amount) {
        acc.balance += amount;
        writeTransaction(acc, "Deposit", amount);
        System.out.println("Successfully deposited " + amount + ". New Balance: " +
acc.balance);
    } void withdraw(Account acc, double amount) {
        if (amount % 100 != 0) {
            System.out.println("Amount should be in multiples of 100!");
            return;
        }
        if (acc.balance >= amount) {
            acc.balance -= amount;
            writeTransaction(acc, "Withdrawal", amount);
            System.out.println("Withdrawal successful! Remaining Balance: " + acc.balance);
        } else {
            System.out.println("Insufficient Funds!");
        }
    }
```

```
  void checkBalance(Account acc) {
      System.out.println("Current Balance: " + acc.balance);
   }
  void transactionHistory(Account acc) {
   String fileName = acc.a_no + ".txt";
      try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
        String line;
         while ((line = reader.readLine()) != null) {
             System.out.println(line);
         }
      } catch (IOException e) {
         System.out.println("Error reading transaction history.");
      }
   }
 private void writeTransaction(Account acc, String type, double amount) {
      String fileName = acc.a_no + ".txt";
      try (FileWriter fileWriter = new FileWriter(fileName, true);
          BufferedWriter bufferedWriter = new BufferedWriter(fileWriter)) {
         bufferedWriter.write(type + ": " + amount + "\n");
         bufferedWriter.write("Date: " + new Date() + "\n");
         bufferedWriter.write("Remaining Balance: " + acc.balance + "\n\n");
      } catch (IOException e) {
         System.out.println("Error writing transaction history.");
      }
   }
}
public class Main {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      ATM atm = new ATM();
```

```
    Account acc = new Account();
    acc.createAcc();
while (true) {
        System.out.println("\nWelcome to ATM");
        System.out.println("1. Deposit Money");
        System.out.println("2. Withdraw Money");
        System.out.println("3. Transaction History");
        System.out.println("4. Check Balance");
        System.out.println("5. Exit");
        System.out.print("Choose an option: ");
        int choice = in.nextInt();
       switch (choice) {
           case 1:
               System.out.print("Enter deposit amount: ");
               double depositAmount = in.nextDouble();
               atm.deposit(acc, depositAmount);
               break;
           case 2:
               System.out.print("Enter withdrawal amount: ");
               double withdrawAmount = in.nextDouble();
               atm.withdraw(acc, withdrawAmount);
               break;
           case 3:
               System.out.println("Transaction History:");
               atm.transactionHistory(acc);
               break;
           case 4:
               atm.checkBalance(acc);
               break;
           case 5:
                System.out.println("Thank you for banking with us!");
                System.exit(0);
                break;
            default:
                System.out.println("Invalid choice. Try again.");
        }
      }
   }
}
```

Output:

```
Enter account holder name:
sweety
Enter Username:
sweety05
Enter a 4-digit PIN:
1234
Enter initial deposit amount: 1000
Congratulations! Account Successfully Created.
Account Details:
Account Number: 1111
Account Holder Name: sweety
Balance: 1000.0

Welcome to ATM
1. Deposit Money
2. Withdraw Money
3. Transaction History
4. Check Balance
5. Exit
```

```
Choose an option: 1
Enter deposit amount: 700
Successfully deposited 700.0. New Balance: 1700.0

Welcome to ATM
1. Deposit Money
2. Withdraw Money
3. Transaction History
4. Check Balance
5. Exit
Choose an option: 2
Enter withdrawal amount: 500
Withdrawal successful! Remaining Balance: 1200.0

Welcome to ATM
1. Deposit Money
2. Withdraw Money
3. Transaction History
4. Check Balance
5. Exit
Choose an option: 3
Transaction History:
Account Created
Account Number: 1111
USER ID: sweety05
Account Holder Name: sweety
PIN: 1234
Balance: 1000.0
Date: Sun Mar 09 17:34:36 GMT 2025
```

```
Deposit: 700.0
Date: Sun Mar 09 17:35:04 GMT 2025
Remaining Balance: 1700.0
Withdrawal: 500.0
Date: Sun Mar 09 17:35:11 GMT 2025
Remaining Balance: 1200.0

Welcome to ATM
1. Deposit Money
2. Withdraw Money
3. Transaction History
4. Check Balance
5. Exit
Choose an option: 4
Current Balance: 1200.0

Welcome to ATM
1. Deposit Money
2. Withdraw Money
3. Transaction History
4. Check Balance
5. Exit
Choose an option: 5
Thank you for banking with us!
```