**📄 ATM Interface Project Documentation**

**📌 Project Title**

**ATM Interface (Console-based Java Application)**

**📋 Objective**

To simulate a simple ATM system using Java with user login, transaction management, and basic banking operations like withdraw, deposit, transfer, and transaction history.

**✅ Features Implemented**

1. **User Authentication** – via User ID and PIN
2. **Transaction History** – records all operations per user
3. **Withdraw** – allows withdrawal with balance check
4. **Deposit** – adds money to account
5. **Transfer** – transfer between two registered users
6. **Quit** – exits the ATM session
7. **Predefined Users** – system has preloaded user data

**🧱 Class Structure**

**1. ATM (Main Class)**

* **Role**: Entry point of the application
* **Responsibilities**:
  + User login and authentication
  + Displaying menu options
  + Handling user choices and routing to respective operations

**2. User**

* **Role**: Represents an individual bank user
* **Fields**:
  + userID – Unique identifier
  + pin – 4-digit security PIN
  + balance – Current account balance
  + TransactionHistory history – User's transaction logs
* **Key Methods**:
  + validatePin(int pin)
  + deposit(int amount)
  + withdraw(int amount)
  + transfer(User recipient, int amount)
  + showBalance()
  + showHistory()

**3. Transaction**

* **Role**: Represents a single transaction (deposit, withdraw, transfer)
* **Fields**:
  + type – Type of transaction (Withdraw/Deposit/Transfer)
  + amount – Amount of transaction
  + recipient – Recipient’s userID (used in transfer)
* **toString()** – Returns readable format of the transaction

**4. TransactionHistory**

* **Role**: Maintains a list of Transaction objects for a user
* **Fields**:
  + List<Transaction> transactions
* **Methods**:
  + addTransaction(Transaction t)
  + showHistory()

**5. Bank**

* **Role**: Acts as a backend to store and manage users
* **Fields**:
  + Map<String, User> users
* **Methods**:
  + authenticate(String userID, int pin)
  + getUser(String userID)

**🧪 How It Works**

**💡 Startup**

* The program welcomes the user.
* Prompts for User ID and PIN.

**🔐 Authentication**

* Validates the user against predefined users in Bank.
* If valid, user enters the ATM system.

**🧭 Menu Options**

User can perform:

1. **Transaction History** – shows a list of previous transactions
2. **Withdraw** – deducts amount after checking for sufficient balance
3. **Deposit** – adds amount to user account
4. **Transfer** – sends money to another registered user
5. **Quit** – ends the session

**👨‍💻 Sample Users**

| **User ID** | **PIN** | **Balance** |
| --- | --- | --- |
| user1 | 1234 | ₹10,000 |
| user2 | 5678 | ₹8,000 |

**Algorithm:**

1. **Initialization**:
   * Create a Scanner object for user input.
   * Initialize a Bank object, which will preload the sample users and their data.
   * Declare a User object, currentUser, to store the authenticated user.
2. **User Authentication Loop**:
   * Start a loop that continues until a user successfully logs in.
   * Prompt the user to enter their **User ID** and **PIN**.
   * Call the bank.authenticate() method with the entered credentials.
   * If authentication is successful, the currentUser object is assigned and the loop is exited.
   * If authentication fails, display an "Invalid credentials" message and prompt the user to try again.
3. **ATM Menu Loop**:
   * Once logged in, start a do-while loop that displays the main ATM menu.
   * The menu offers five options:
     1. Transaction History
     2. Withdraw
     3. Deposit
     4. Transfer
     5. Quit
   * Prompt the user to enter their choice.
4. **Handle User's Choice (Switch Statement)**:
   * Use a switch statement to execute the corresponding action based on the user's choice.
   * **Case 1 (Transaction History)**: Call currentUser.showHistory() to display all recorded transactions for the current user.
   * **Case 2 (Withdraw)**:
     1. Prompt for the amount to withdraw.
     2. Call currentUser.withdraw() with the amount.
     3. The withdraw method will handle balance checks and transaction logging.
   * **Case 3 (Deposit)**:
     1. Prompt for the amount to deposit.
     2. Call currentUser.deposit() with the amount.
     3. The deposit method will update the balance and log the transaction.
   * **Case 4 (Transfer)**:
     1. Prompt for the recipient's User ID.
     2. Retrieve the recipient's User object from the Bank using bank.getUser().
     3. If the recipient is found, prompt for the transfer amount and call currentUser.transfer() to complete the transaction, which updates both user balances and logs the transfer for both parties.
     4. If the recipient is not found, display an error message.
   * **Case 5 (Quit)**: Exit the do-while loop and display a goodbye message.
   * **Default**: For any other input, display an "Invalid option" message.
5. **Termination**:
   * The program exits the menu loop when the user chooses '5'.
   * Close the Scanner object.
   * Display a final thank you message.

**Program Code:**

import java.util.\*;

// Class 1: Transaction

class Transaction {

String type;

int amount;

String recipient;

Transaction(String type, int amount) {

this.type = type;

this.amount = amount;

}

Transaction(String type, int amount, String recipient) {

this.type = type;

this.amount = amount;

this.recipient = recipient;

}

public String toString() {

if (recipient != null) {

return type + " of Rs." + amount + " to " + recipient;

}

return type + " of Rs." + amount;

}

}

// Class 2: TransactionHistory

class TransactionHistory {

private List<Transaction> transactions;

TransactionHistory() {

transactions = new ArrayList<>();

}

void addTransaction(Transaction t) {

transactions.add(t);

}

void showHistory() {

if (transactions.isEmpty()) {

System.out.println("No transactions yet.");

} else {

System.out.println("Transaction History:");

for (Transaction t : transactions) {

System.out.println("- " + t);

}

}

}

}

// Class 3: User

class User {

String userID;

int pin;

int balance;

TransactionHistory history;

User(String userID, int pin, int balance) {

this.userID = userID;

this.pin = pin;

this.balance = balance;

this.history = new TransactionHistory();

}

boolean validatePin(int inputPin) {

return this.pin == inputPin;

}

void deposit(int amount) {

balance += amount;

System.out.println("Successfully deposited Rs." + amount);

history.addTransaction(new Transaction("Deposit", amount));

}

boolean withdraw(int amount) {

if (amount <= balance) {

balance -= amount;

System.out.println("Please collect your cash.");

history.addTransaction(new Transaction("Withdraw", amount));

return true;

} else {

System.out.println("Insufficient balance.");

return false;

}

}

boolean transfer(User recipient, int amount) {

if (amount <= balance) {

balance -= amount;

recipient.balance += amount;

history.addTransaction(new Transaction("Transfer", amount, recipient.userID));

recipient.history.addTransaction(new Transaction("Received", amount, this.userID));

System.out.println("Successfully transferred Rs." + amount + " to " + recipient.userID);

return true;

} else {

System.out.println("Insufficient balance.");

return false;

}

}

void showBalance() {

System.out.println("Current Balance: Rs." + balance);

}

void showHistory() {

history.showHistory();

}

}

// Class 4: Bank

class Bank {

private Map<String, User> users;

Bank() {

users = new HashMap<>();

// Preloaded users

users.put("user1", new User("user1", 1234, 10000));

users.put("user2", new User("user2", 5678, 8000));

}

User authenticate(String userID, int pin) {

User user = users.get(userID);

if (user != null && user.validatePin(pin)) {

return user;

}

return null;

}

User getUser(String userID) {

return users.get(userID);

}

}

// Class 5: ATM (Main Class)

public class ATM {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Bank bank = new Bank();

System.out.println("==== Welcome to the ATM System ====");

// Login Loop

User currentUser = null;

while (currentUser == null) {

System.out.print("Enter User ID: ");

String userID = sc.nextLine();

System.out.print("Enter PIN: ");

int pin = sc.nextInt();

sc.nextLine(); // consume newline

currentUser = bank.authenticate(userID, pin);

if (currentUser == null) {

System.out.println("Invalid User ID or PIN. Please try again.\n");

}

}

System.out.println("\nLogin successful! Welcome " + currentUser.userID + "\n");

// ATM Menu

int choice = 0;

do {

System.out.println("====== ATM Menu ======");

System.out.println("1. Transaction History");

System.out.println("2. Withdraw");

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

System.out.println("4. Transfer");

System.out.println("5. Quit");

System.out.print("Enter your choice: ");

choice = sc.nextInt();

sc.nextLine(); // consume newline

switch (choice) {

case 1:

currentUser.showHistory();

break;

case 2:

System.out.print("Enter amount to withdraw: ");

int withdrawAmount = sc.nextInt();

sc.nextLine();

currentUser.withdraw(withdrawAmount);

break;

case 3:

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

int depositAmount = sc.nextInt();

sc.nextLine();

currentUser.deposit(depositAmount);

break;

case 4:

System.out.print("Enter recipient User ID: ");

String recipientID = sc.nextLine();

User recipient = bank.getUser(recipientID);

if (recipient == null) {

System.out.println("Recipient not found.");

} else {

System.out.print("Enter amount to transfer: ");

int transferAmount = sc.nextInt();

sc.nextLine();

currentUser.transfer(recipient, transferAmount);

}

break;

case 5:

System.out.println("Thank you for using the ATM. Goodbye!");

break;

default:

System.out.println("Invalid option. Try again.");

}

System.out.println();

} while (choice != 5);

sc.close();

}

}

**📚 Conclusion**

This Java project demonstrates a functional and modular **ATM interface** with authentication and banking features using **OOP principles**. It's ideal for learning about:

* Classes & Objects
* Collections (List, Map)
* User Input Handling
* Program Structure and Flow Control