

ATM Interface

A Simulation of Banking Operations Using Java

PRESENTED BY

Logeshwari N

Tagore Engineering College

Enrollment Number : EBEON03251078045

Batch : 2024-14789

Introduction

- Objective: To develop a console-based ATM application that simulates basic banking operations.
- Purpose: Demonstrate the application of object-oriented programming (OOP) principles in Java.
- Features:
- User authentication via ID and PIN.
- Perform transactions: Deposit, Withdraw, Transfer.
- View transaction history.



Tools and Technologies Used

Programming Language: Java
Development Environment: Visual Studio Code (VS Code)
Java Development Kit (JDK): JDK 24
Extensions and Tools:
Java Extension Pack (includes Language Support, Debugger, Maven, etc.)
Java Language Pack (for UI localization)
Build Tools: Maven or Gradle (for project management and build automation)
Version Control: Git (optional, for source code management)



Project Architecture

The architecture of the ATM Interface project follows a modular, object-oriented structure. The system is designed to separate different functionalities into distinct classes, making the project easy to understand, manage, and scale. Each class performs a specific role in the ATM system.

A

Class Structure:
AccountHolder: Stores user credentials.
Account: Manages account balance.
BankTransaction: Records transaction details.
Bank: Handles core banking operations.
ATM: User interface for interaction.

B

Design Principles:
Encapsulation: Each class has specific responsibilities.
Modularity: Separation of concerns for maintainability



Class Descriptions



AccountHolder:

Attributes: userID, userPIN.
Methods: validateCredentials().

Account:

Attributes: balance.
Methods: deposit(), withdraw(), getBalance()

BankTransaction:

Attributes: transactionType, amount, date.
Methods: getTransactionDetails().

Bank:

Attributes: AccountHolder, Account, List of BankTransaction.
Methods: authenticate(), deposit(), withdraw(), transfer(), showTransactionHistory().

ATM:

Role: Main class to interact with the user and invoke Bank methods.

User Authentication

Process:

- Prompt user for ID and PIN.
- Validate credentials using AccountHolder class.
- Grant or deny access based on validation.

Security Considerations:

- Simple authentication for demonstration purposes.
- In real-world applications, implement secure authentication mechanisms.

CODING

The screenshot shows a Java development environment with the following details:

- Toolbar:** Includes icons for file operations (New, Open, Save, Print, Find, Copy, Paste, Cut, Undo, Redo), search, terminal, help, and system status.
- Header:** "Search the web" and "ATMProject".
- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** "Search the web" and "ATMProject".
- Explorer:** Shows the project structure under "ATMPROJECT": Account.class, Account.java, AccountHolder.class, AccountHolder.java, ATM.class, ATM.java (selected), Bank.class, Bank.java, BankTransaction.class, and BankTransaction.java.
- Code Editor:** Displays the content of ATM.java. The code initializes a Bank object and a Scanner, then enters a loop to prompt the user for a choice from a menu of 1 to 5. It handles withdrawal requests by printing the amount and checking if the bank can withdraw it. If successful, it prints a withdrawal message; otherwise, it prints an error message.

```
import java.util.Scanner;
public class ATM {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Bank bank = new Bank(userId:"user123", userPin:"1234");
        System.out.print("Enter User ID: ");
        String userId = scanner.nextLine();
        System.out.print("Enter User PIN: ");
        String userPin = scanner.nextLine();
        if (bank.authenticate(userId, userPin)) {
            int choice;
            do {
                System.out.println("\nATM Menu:");
                System.out.println("1. Show 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 = scanner.nextInt();
                switch (choice) {
                    case 1:
                        bank.showTransactionHistory();
                        break;
                    case 2:
                        System.out.print("Enter amount to withdraw: ");
                        double withdrawAmount = scanner.nextDouble();
                        if (bank.withdraw(withdrawAmount)) {
                            System.out.println("Withdrawal successful.");
                        } else {
                            System.out.println("Insufficient funds or error during withdrawal.");
                        }
                }
            } while (choice != 5);
        } else {
            System.out.println("Authentication failed. Please try again.");
        }
    }
}
```

- Bottom Status Bar:** Shows "Java: Ready", "Ln 67, Col 2", "Spaces: 4", "UTF-8", "LF", "Java", "Go Live", "Prettier", and system status like temperature (36°C), weather (Mostly sunny), and date/time (04-05-2025).

A screenshot of a Windows desktop environment showing a Java project in a code editor. The project is named "ATMProject" and contains several Java files: Account.class, Account.java, AccountHolder.class, AccountHolder.java, ATM.class, ATM.java, Bank.class, Bank.java, BankTransaction.class, and BankTransaction.java. The file "Bank.java" is currently open and selected in the Explorer sidebar.

The code in "Bank.java" is as follows:

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class Bank {
5     private AccountHolder holder;
6     private Account account;
7     private List<BankTransaction> transactions;
8
9     public Bank(String userId, String userPin) {
10         holder = new AccountHolder(userId, userPin);
11         account = new Account();
12         transactions = new ArrayList<>();
13     }
14
15     public boolean authenticate(String userId, String userPin) {
16         return holder.getUserId().equals(userId) && holder.getUserPin().equals(userPin);
17     }
18
19     public void deposit(double amount) {
20         account.deposit(amount);
21         transactions.add(new BankTransaction(type:"Deposit", amount));
22     }
23
24     public boolean withdraw(double amount) {
25         if (account.withdraw(amount)) {
26             transactions.add(new BankTransaction(type:"Withdraw", amount));
27             return true;
28         }
29         return false;
30     }
31
32     public boolean transfer(Bank otherBank, double amount) {
33         if (account.transfer(otherBank.account, amount)) {
34             transactions.add(new BankTransaction(type:"Transfer", amount));
35         }
36     }
37 }
```

The status bar at the bottom shows "Java: Ready", "Ln 1, Col 1", "Spaces: 4", "UTF-8", "LF", "Java", "Go Live", and "Prettier". The system tray indicates it's 36°C and mostly sunny, with a battery level of 15% and the date/time as 04-05-20.

A screenshot of a Windows desktop environment. In the foreground, a Java code editor window is open, displaying the code for `BankTransaction.java`. The code defines a class `BankTransaction` with private fields `type`, `amount`, and `date`, and methods `BankTransaction(String type, double amount)` and `String toString()`. The code editor interface includes a top bar with file icons, a search bar, and a tab bar showing `BankTransaction.java` and `Bank.java`. The left sidebar shows a project tree for `ATMPROJECT` containing files like `Account.class`, `Account.java`, etc. The bottom status bar shows Java: Ready, file statistics (Ln 1, Col 1, Spaces: 4, UTF-8, LF), and various system icons.

```
import java.util.Date;  
public class BankTransaction {  
    private String type;  
    private double amount;  
    private Date date;  
  
    public BankTransaction(String type, double amount) {  
        this.type = type;  
        this.amount = amount;  
        this.date = new Date();  
    }  
  
    public String toString() {  
        return date + " - " + type + ": $" + amount;  
    }  
}
```

The screenshot shows a Java development environment with the following details:

- Toolbar:** Includes icons for file operations, search, and various tools.
- Search Bar:** Displays "Search the web" and a search icon.
- File Menu:** Contains File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Displays "ATMProject" and a refresh icon.
- Editor Area:** Shows the code for `AccountHolder.java`.

```
public class AccountHolder {
    private String userId;
    private String userPin;

    public AccountHolder(String userId, String userPin) {
        this.userId = userId;
        this.userPin = userPin;
    }

    public String getUserId() {
        return userId;
    }

    public String getUserPin() {
        return userPin;
    }
}
```
- Explorer View:** Shows the project structure under "ATMPROJECT". The file `AccountHolder.java` is selected.
- Sidebar:** Includes sections for OUTLINE, TIMELINE, and JAVA PROJECTS.
- Bottom Status Bar:** Shows "Java: Ready", "Ln 1, Col 1", "Spaces: 4", "UTF-8", "LF", "Java", "Go Live", "Prettier", and system status like battery level and temperature.
- Taskbar:** Shows the Start button, Search bar, File Explorer, Task View, File History, Mail, Microsoft Edge, Google Chrome, and the VS Code icon.

OUTPUT

The screenshot shows a terminal window in a dark-themed code editor interface. The terminal tab is active, displaying the output of a Java application named ATMProject. The output shows the compilation of source files and the execution of the application, which prompts for User ID and PIN, displays an ATM menu, and performs withdrawal, deposit, and transfer operations.

```
PS C:\Users\Logeshwari\Downloads\ATMProject> javac *.java
PS C:\Users\Logeshwari\Downloads\ATMProject> java ATM
Enter User ID: user123
Enter User PIN: 1234

ATM Menu:
1. Show Transaction History
2. Withdraw
3. Deposit
4. Transfer
5. Quit
Enter your choice: 1
No transactions yet.

ATM Menu:
1. Show Transaction History
2. Withdraw
3. Deposit
4. Transfer
5. Quit
Enter your choice: 2
Enter amount to withdraw: 1000
Insufficient balance.

ATM Menu:
1. Show Transaction History
2. Withdraw
3. Deposit
4. Transfer
5. Quit
Enter your choice: 4
Enter amount to transfer: 200
Transfer failed.

ATM Menu:
1. Show Transaction History
2. Withdraw
3. Deposit
```

At the bottom of the terminal window, there is a status bar with various icons and text, including "Java: Ready", "Ln 67, Col 2", and "Spaces: 4". The bottom right corner shows the date and time as "04-05-2025 15:55".

Conclusion

- Successfully simulated a working ATM interface using Java.
- Effectively applied Object-Oriented Programming (OOP) principles like encapsulation and modularity.
- Gained hands-on experience with console-based user interaction and structured application logic.
- Integrate persistent storage using a database (e.g., MySQL) to store user data and transaction history.
- Upgrade from console-based UI to a Graphical User Interface (GUI) using JavaFX or Swing.



Thank you!

**This project helped me learn Java and
build a working ATM system.**