

## Design Overview for Console ATM

Name: Luong Trac Duc Anh

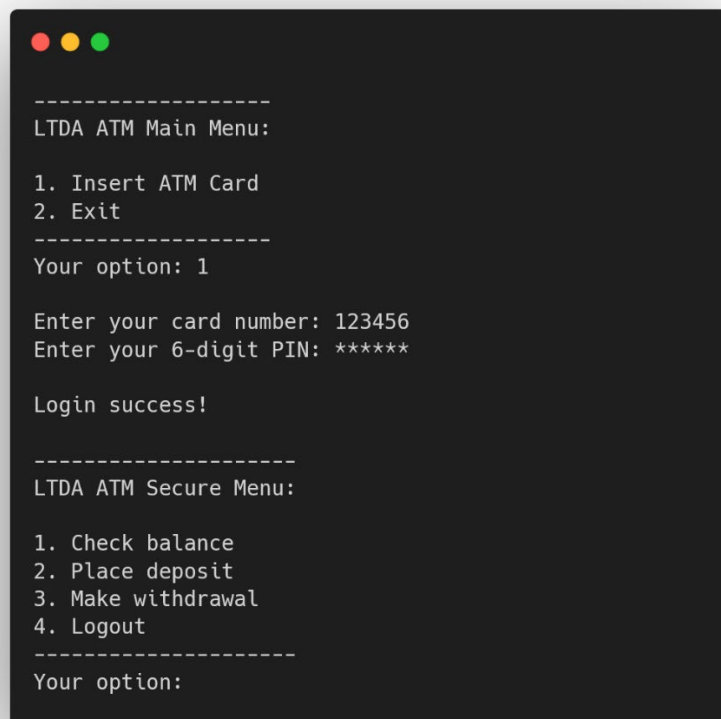
Student ID: 103488117

### Summary of Program

This program will mimic the functionalities of a real-life ATM, showing the main menu to insert your ATM card and log in via a 6-digit PIN. After you have successfully logged in, a secure menu will show up with 4 options: Check balance, Place deposit, Make withdrawal and Logout.

- **Check balance** will print the current money amount you have on your ATM card to the screen.
- **Place deposit** will allow you to insert an amount of money (up to your choice) into the account, and your balance will also increment with that number, starting from the initial that the program run. After that, the program will display your current balance number.
- **Make withdrawal** will allow you to cash out an amount of money (up to your choice but has to be within the account's current balance). Next, your balance will also decrease based on the amount of money that you have drawn. After that, the program will display your current balance number.
- **Logout** will exit your account and return to the main screen, with 2 options: Insert ATM Card or Exit.

Sketch:



```
-----
LTDA ATM Main Menu:

1. Insert ATM Card
2. Exit
-----
Your option: 1

Enter your card number: 123456
Enter your 6-digit PIN: ****

Login success!

-----
LTDA ATM Secure Menu:

1. Check balance
2. Place deposit
3. Make withdrawal
4. Logout
-----
Your option:
```

## Added Complexity

For the High Distinction standard, I have added more complexity to the Console ATM program, which includes the implementation of OOP concepts (encapsulation and abstraction) and different design patterns which will be fully described in the next section (Design Patterns).

### New features

- 2 more options in the Secure Menu: Money Transfer (use to send money within the bank's system) and Transactions (use to view transaction history)
- 6 interfaces in correlation to 6 features in the secure menu
- 2 enumerations (TransactionType and SecureScreen)
- 2 Lists (BankAccount and Transaction)
- Utility class, POCO class (Plain Old CLR Objects)
- Static fields and static classes
- "var" and "const" keyword
- Guard clauses (instead of nested if statements)
- While and foreach loops
- CultureInfo classes (using System.Globalization) to get currency symbols (I tried to use Vietnamese currency but could not fix the error, therefore I am using Australian currency (AUD) instead)

## Design Patterns

### Chain of Responsibility

You can pass requests along a chain of handlers using the behavioural design pattern known as Chain of Responsibility. Each handler chooses whether to execute a request or pass it on to the handler behind it in the chain.

In real-life situations, when you deposit an amount of cash in the ATM, it will just count every individual banknote and give you a sum preview before confirming your transaction. However, in the program, on a console, you can only input a number and let the machine calculate the number of banknotes and their values using division and continue the loop with the remainder, and so on. There will be a guard clause that only allows you to insert an amount that is a multiplication of 10. I can implement the chain of responsibility pattern with the 3 handler methods like FiftyHandler(), TwentyHandler(), and TenHandler() when the PlaceDeposit() method is used. This will result in a chain of counting banknotes with the value of 50, 20, and 10 AUD.

### State

Another behavioural design pattern called state enables objects to change their behaviour as their internal states change. It appears as if the object's class has changed.

This design pattern can be integrated into the program via Ilogin, Ibalance, and Itransaction interfaces.

- For the Ilogin interface, you will have 3 attempts to input the correct pin code to your ATM card. If you failed all 3 attempts, the state of your BankAccount will be changed to locked, and the bool value IsLocked will be changed to true.
- For the Ibalance interface, each time you execute a transaction, the property Balance will be changed according to whether you choose to deposit or withdraw money.
- For the Itransaction interface, every time you carry out a transaction, it will append a transaction to the \_listOfTransactions. If it is a money transfer, the constructor will call the VMTransfer() class and send the money to the recipient account.

## Advancement in comparison to the Distinction Task

With added complexity and design pattern well implemented, I believe the program qualifies for the High Distinction standards.

The program separates different features and organises them by using interfaces, which then can be implemented in different objects and classes. The procedure for the program (the sequence diagram will be included for submission) is designed close to a real ATM, for example, your account will be locked after 3 wrong PIN code inputs during the login process.

With guard clauses in correspondence with each function, the program is less prone to errors. This is an important note as the program is a console application, which will operate entirely based on the user's inputs. With the help of GetInput methods() defined in the Utility class, each step that involves the user's decision based on the number they entered will be closely validated. Every error will be printed on the screen, with green for confirmation and red if an error occurs.

The depth of research will be demonstrated in the code, documentation and also usage of external packages. For the View Transactions option, I have implemented the ConsoleTables package (more information via this link: <https://www.nuget.org/packages/ConsoleTables>, the version that I'm currently using is 2.4.2). This allows me to create tables that display the transaction history, to increase readability and ease of use for the end user.

## Required Roles

Table 1: <<role name>> details – duplicate

Responsibility	Type Details	Notes
<b>Name</b>	string	Property
<b>AccountNumber</b>	Int64	Property
<b>CardNumber</b>	Int64	Property
<b>PinCode</b>	Int64	Property
<b>Balance</b>	decimal	Property
<b>IsLocked</b>	bool	Property
<b>CheckBalance()</b>	void	Check the account's current balance amount

<b>PlaceDeposit()</b>	void	Insert money into the user's account
<b>CheckPassword()</b>	void	Check the user's entered pin code with the BankAccount list
<b>InsertTransaction()</b>	void	Add the transaction to the list of transaction
<b>ViewTransaction()</b>	void	Display the transaction history
<b>PerformTransfer()</b>	void	Send money from one bank account to another
<b>MakeWithdrawal()</b>	void	Cash out a sufficient amount of money from the entered account
<b>attempts</b>	int	Number of times you try to login
<b>maxAttempts</b>	int	Set the max number of attempts to 3 before your account has to be locked
<b>minimumBalance</b>	int	Set the maintenance amount of money in your account to 10 AUD
<b>transactionAmount</b>	decimal	To validate withdrawal and transfer amount with the account balance
<b>_accountList</b>	List	Create the initial bank accounts
<b>_listOfTransaction</b>	List	Used for the InsertTransaction() and ViewTransaction() methods
<b>selectedAccount</b>	field	The account number after login successfully
<b>inputAccount</b>	field	The entered account number when login
<b>Initialize()</b>	void	Create initial bank accounts
<b>Execute()</b>	void	Do actions displayed in the Main Menu and Secure Menu
<b>LockAccount()</b>	void	Terminate the process and return an exit code
<b>PreviewBankNotes()</b>	void	Display the banknote values that the user inserted

Table 2: <<enumeration name>> details

Value	Notes
<b>SecureMenu</b>	Predefines the description and assign the option for the switch statement based on the user's input
<b>TransactionType</b>	Predefines the transaction types that the ATM can have: Deposit, Withdrawal, Transfer