Assignment 1: ATM Simulator with Function Parameters

Objective

Simulate a simple ATM system that allows a user to check their balance, deposit money, withdraw money, print a mini statement, and exit the system, while demonstrating different **function parameter passing techniques in C#**.

Implementation Steps

1. **Create a new C# Console Application.**
2. **Display your name and the program title** at the top of the program.
    - Display:
      `=== Simple ATM System ===`
3. **Initialize the account balance** to `1000.00`.
    - Display once at the start:
      `Initial Balance: ₱1000.00`
4. **Use a `while` loop** to keep the program running until the user selects the Exit option.
5. **Display the ATM menu using `switch-case`:**
    - 1: Check Balance
    - 2: Deposit Money
    - 3: Withdraw Money
    - 4: Print Mini Statement
    - 5: Exit

Menu Options and Required Function Usage

Option 1: Check Balance

- Create a function that **receives the balance by value**.
- Display:
  `Current Balance: ₱{amount}`

📌 *Pass-by-value is used because the balance is only read.*

Option 2: Deposit Money

- Prompt:
  `Enter amount to deposit:`
- Validate that the amount is **positive** using `if-else`.
- Create a function that **updates the balance using `ref`**.
- On success:
  `Deposit successful.`
  `Updated Balance: ₱{amount}`
- On invalid input:
  `Invalid deposit amount. Please enter a positive value.`
    - Use `continue`.

📌 *`ref` allows the balance to be modified.*

Option 3: Withdraw Money

- Prompt:
  `Enter amount to withdraw:`
- Create a function that:
    - Uses **`ref`** to update the balance
    - Uses **`out`** to return the withdrawal status
- On success:
  `Withdrawal successful.`
  `Updated Balance: ₱{amount}`
- If insufficient balance:
  `Withdrawal failed. Insufficient balance.`
- If invalid amount:
  `Invalid withdrawal amount. Please enter a positive value.`
    - Use **`continue`**.

📌 *`ref` updates the balance, `out` communicates success or failure.*

Option 4: Print Mini Statement

- Create a function that **receives values by value**.
- Display:

```
--- Mini Statement ---
Current Balance: ₱{amount}
Last Transaction Amount: ₱{amount}
```

📌 *Pass-by-value is used because this option is display-only.*

Option 5: Exit

- Display:

```
Thank you for using the ATM. Goodbye!
```

- Use `break` to exit the program loop.

Input Validation and Control Flow

6. Use **`if-else`** for validating amounts.
7. Use **`continue`** to repeat the menu on invalid input.
8. If the user enters an invalid menu option, display:

```
Invalid option selected. Please try again.
```

    - Redirect the user back to the menu.

📌 Important Implementation Note (Read Before Coding)

For this activity, your program is **already structured**.
You are **NOT allowed to modify `Program.cs`**.

You will work **ONLY** on the following files:

✅ `BankingService.cs`

- This file contains **all business logic**
- Implement all ATM-related operations here:
    - Check Balance (pass-by-value)

    - Deposit (using `ref`)
    - Withdraw (using `ref` and `out`)
    - Store the last transaction amount
    - Logic for printing the bank statement (pass-by-value)
- **Do NOT use `Console.ReadLine()` or `Console.WriteLine()` in this file**
- This file should be **unit-test friendly**

📌 Think of this as *"the brain of the ATM"*

✅ `BankingView.cs`

- This file is responsible for **all user interaction**
- Display menus, prompts, and messages such as:
    - ATM menu options
    - Balance display
    - Error messages
    - Mini statement output

- **No business logic should be written here**

📌 Think of this as *"the screen of the ATM"*

🚫 `Program.cs` (DO NOT MODIFY)

- Controls the program flow:
    - `while` loop
    - `switch-case`
    - `continue` and `break`
- Calls methods from:
    - `BankingService`
    - `BankingView`

📌 If your solution requires changing `Program.cs`, your design is incorrect.