Simple Banking System

Subject: Object-Oriented Programming 1
Student: Mhar John Y. Gerarman
Teacher: Engr. Julian N. Semblante

Project Overview

Purpose: To create a functional, secure, and user-friendly console application that simulates a real-world banking system.

Accomplishments:

- Demonstrated core Object-Oriented Programming (OOP) principles.
- Implemented robust data persistence using flat-file CSVs.
- Achieved a clean Separation of Concerns (Logic and UI).

Key Features:

- Dual user roles: Customer and Employee (Admin).
- Secure user registration and password-masked login.
- Polymorphic account creation (6 account types).
- Full transaction logic (Deposit, Withdraw).
- Administrative-level account management (Search, View All, Delete).

Technology Stack:

- Language: C# (.NET 6+)
- Platform: .NET Console Application
- Storage: CSV (Flat-file database)
- Core Libraries: System.IO, System.Linq, System.Globalization

Requirements & Installation

Software Requirements:

- .NET 6 SDK or newer
- – Any code editor/IDE (Visual Studio 2022 or VS Code)
- – Console/terminal with UTF-8 support for UI characters

System Requirements:

- The application is lightweight and will run on any modern Windows, macOS, or Linux machine where the .NET runtime is installed.

Installation / Running the Program:

- Download or clone the project repository
- Open the folder or .sln file in your chosen IDE
- Run the program (Visual Studio Start button or dotnet run)
- No extra dependencies needed besides the .NET SDK

File Handling Overview

File Types & Purpose

- users.csv – Stores customer information

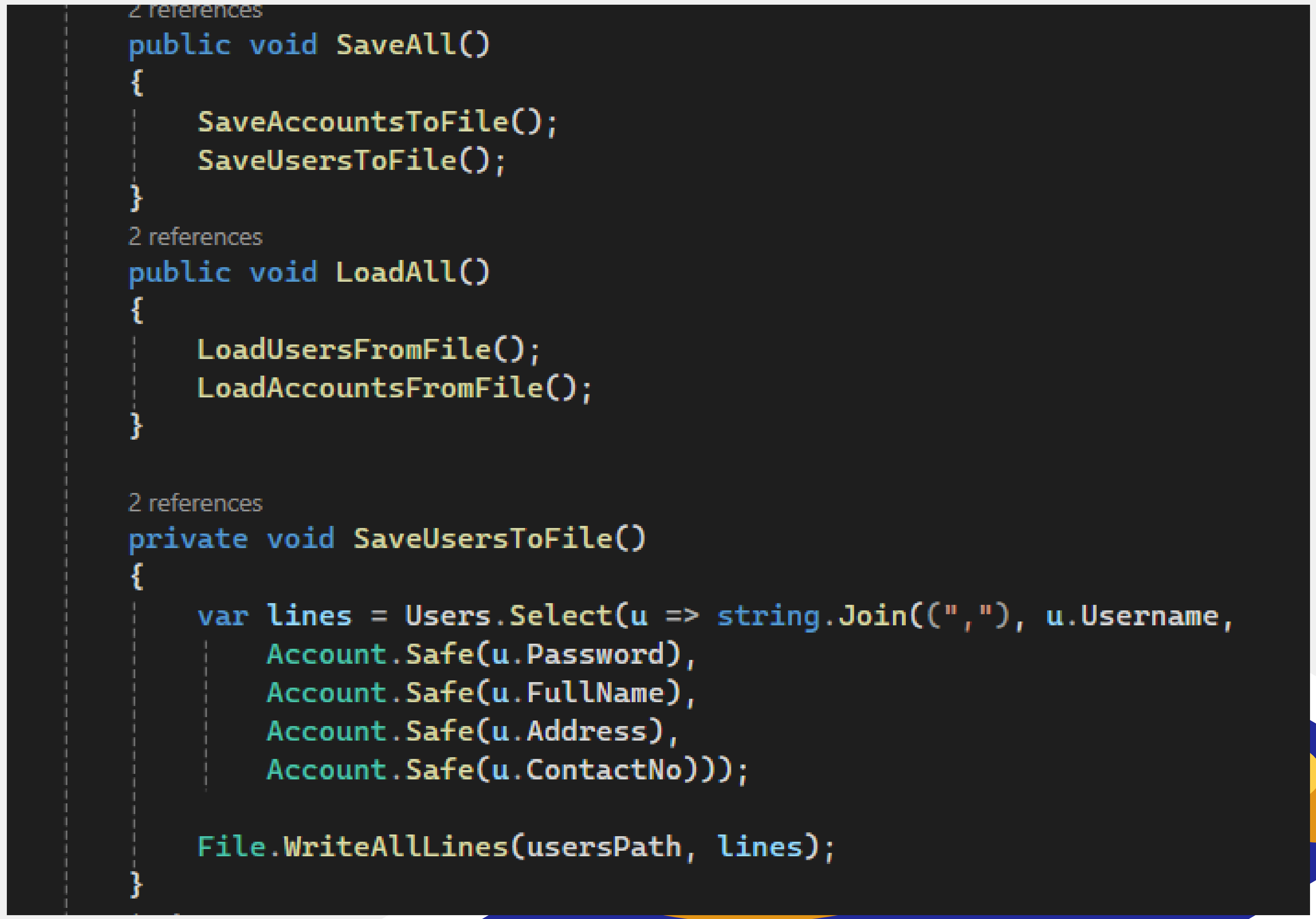
(Username, Password, FullName, Address, ContactNo)

- accounts.csv – Stores bank account details

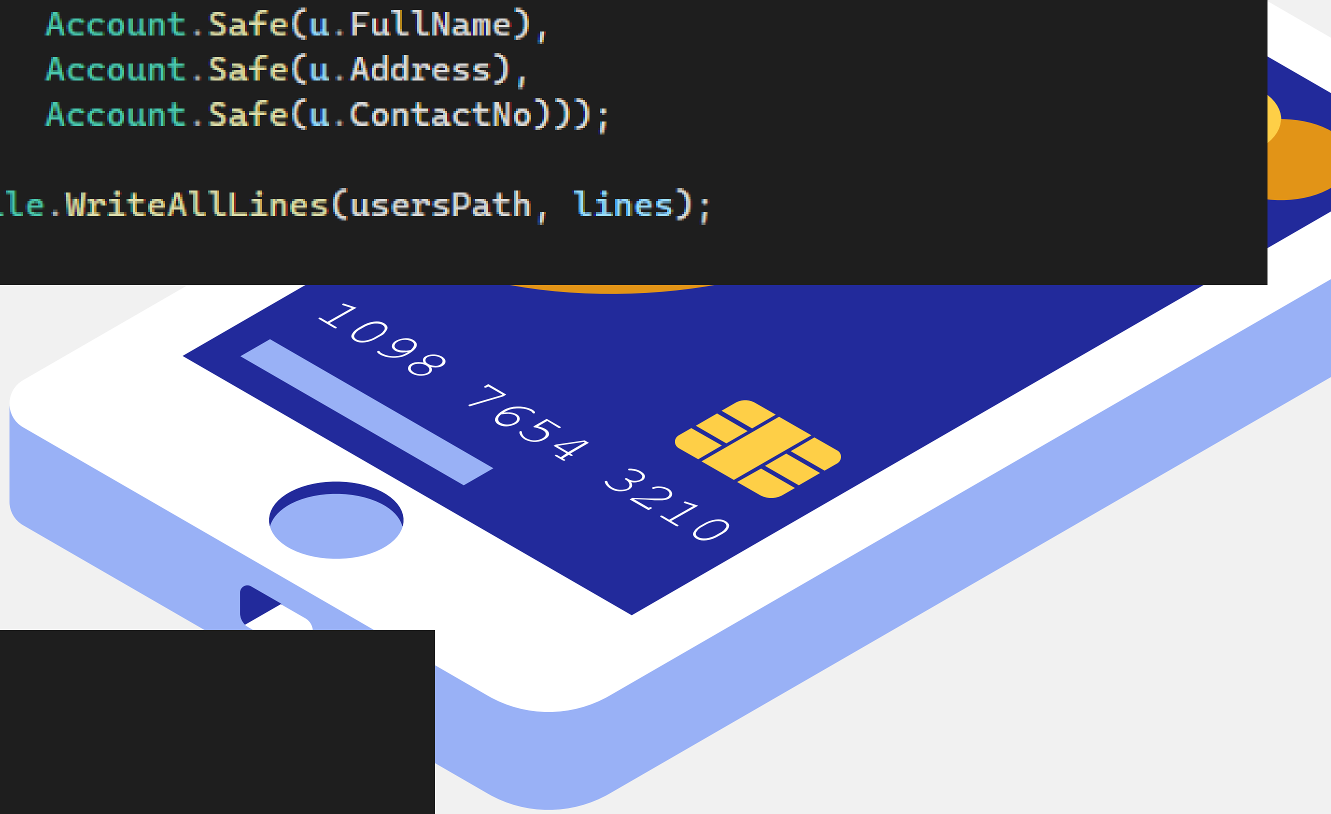
(AccountID, Type, OwnerUsername, Balance, extra fields)

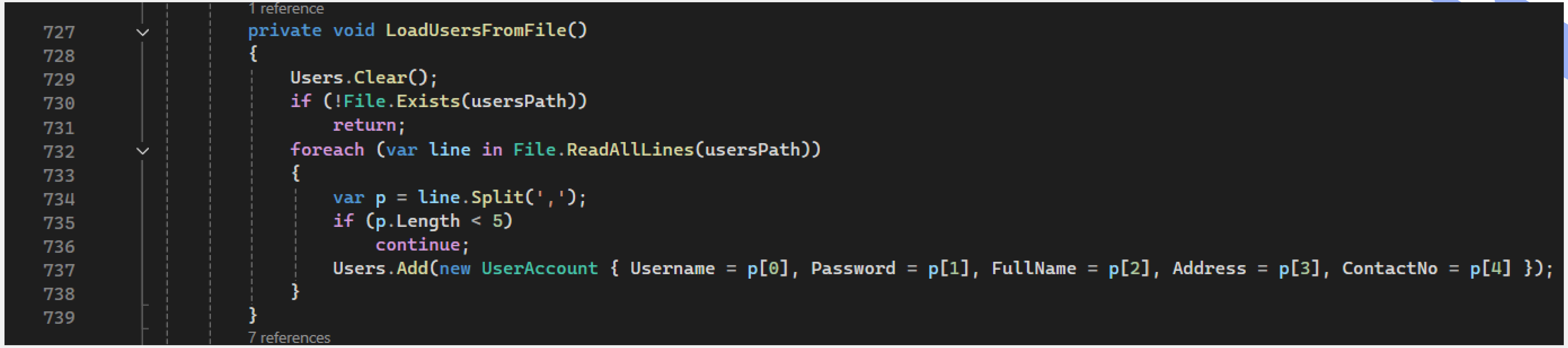
File Operations

- **Read on Startup:** LoadAll() loads both CSV files into memory
- **Write on Update:** SaveAll() overwrites CSVs after every successful change
- Ensures data persistence (data never resets)



```
2 references
public void SaveAll()
{
    SaveAccountsToFile();
    SaveUsersToFile();
}
2 references
public void LoadAll()
{
    LoadUsersFromFile();
    LoadAccountsFromFile();
}

2 references
private void SaveUsersToFile()
{
    var lines = Users.Select(u => string.Join((","), u.Username,
        Account.Safe(u.Password),
        Account.Safe(u.FullName),
        Account.Safe(u.Address),
        Account.Safe(u.ContactNo)));

    File.WriteAllLines(usersPath, lines);
}
```



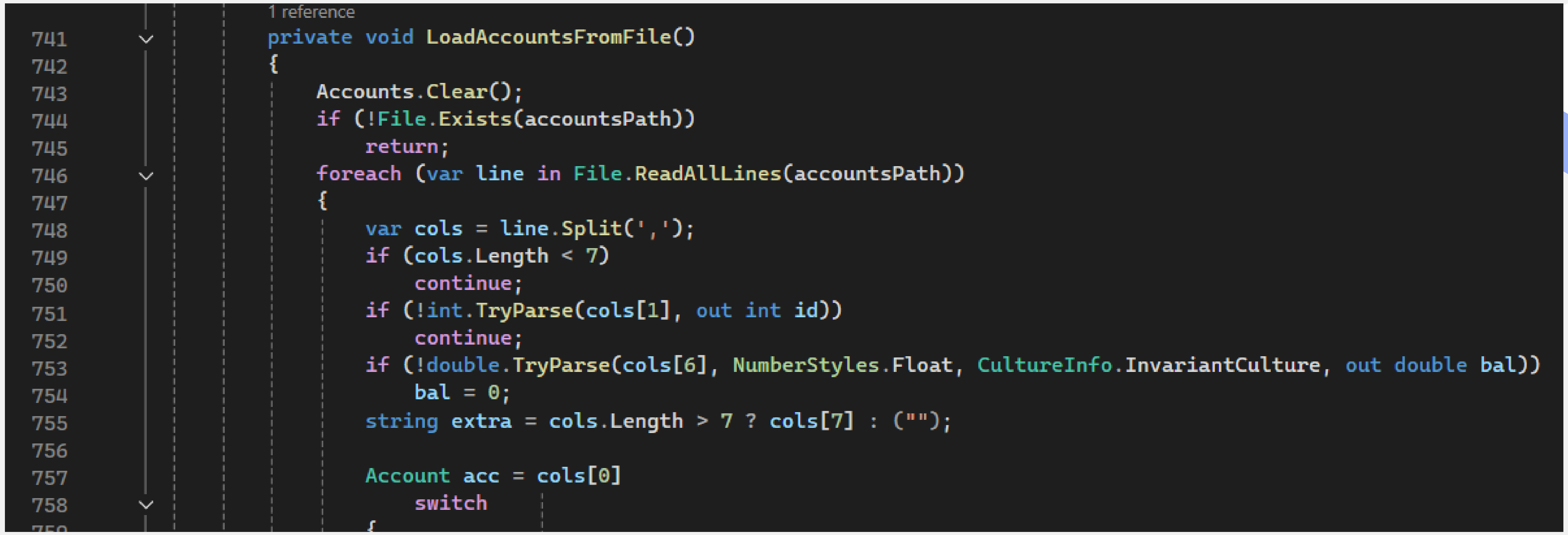


```
1 reference
private void LoadUsersFromFile()
{
    Users.Clear();
    if (!File.Exists(usersPath))
        return;
    foreach (var line in File.ReadAllLines(usersPath))
    {
        var p = line.Split(',');
        if (p.Length < 5)
            continue;
        Users.Add(new UserAccount { Username = p[0], Password = p[1], FullName = p[2], Address = p[3], ContactNo = p[4] });
    }
}
7 references
```

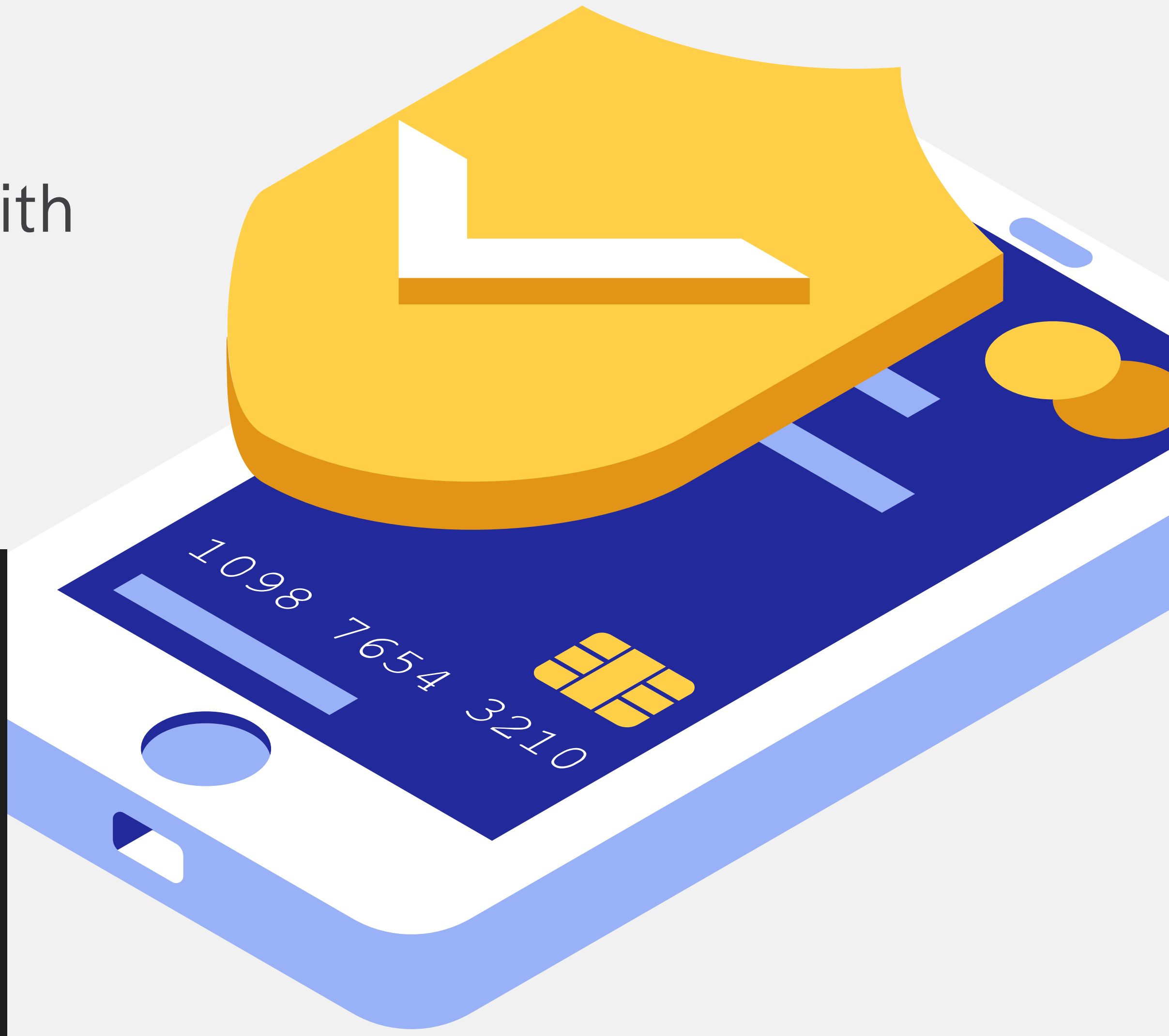
File Handling Overview

Error Handling

- **File Check: File.Exists()** prevents errors when files are missing.
- Uses **double.TryParse** and **int.TryParse** to avoid crashes from invalid data.
- **(Safe() + CultureInfo.InvariantCulture)** ensure correct saving/reading even with commas and different decimal formats.



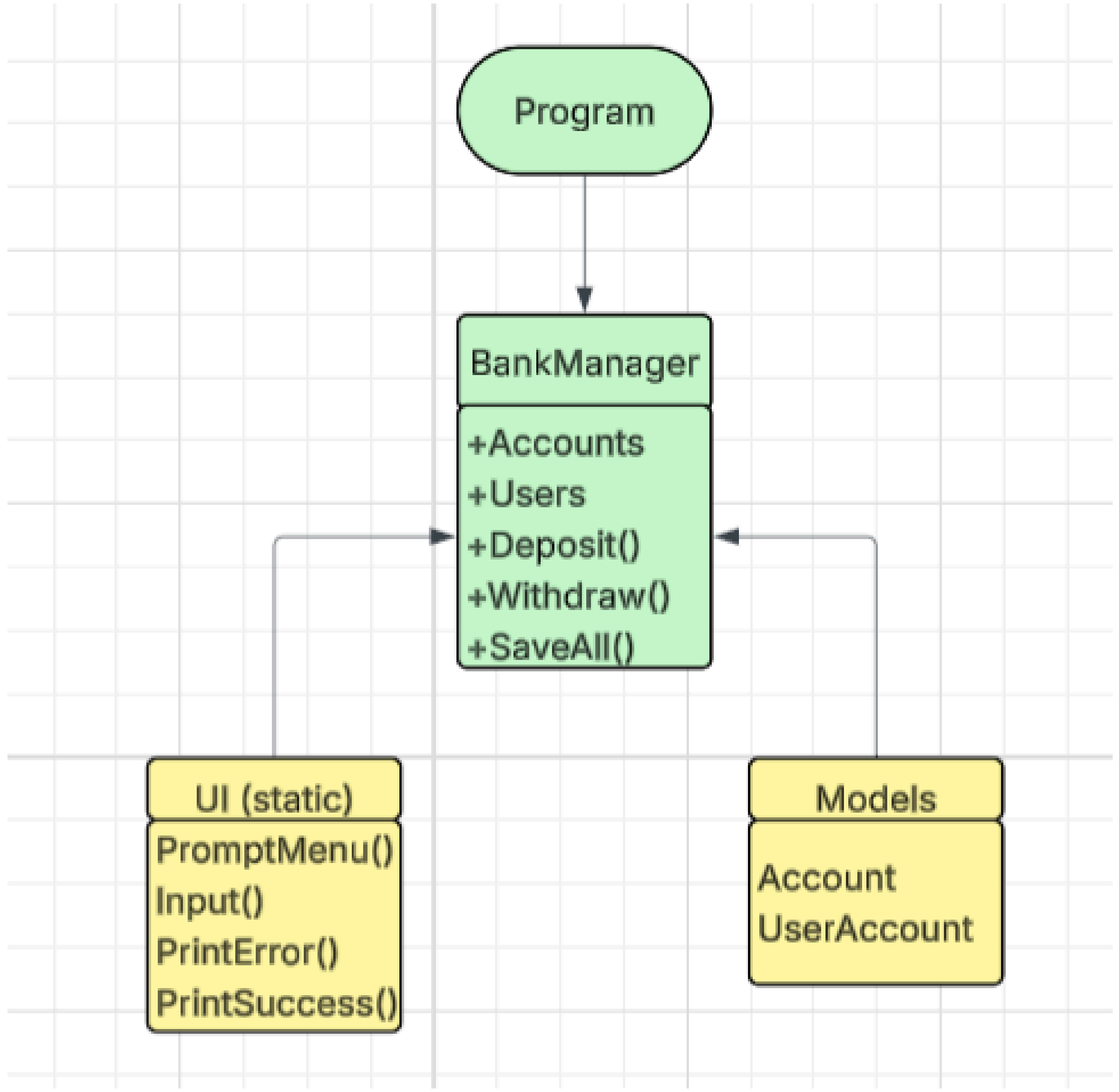
```
1 reference
private void LoadAccountsFromFile()
{
    Accounts.Clear();
    if (!File.Exists(accountsPath))
        return;
    foreach (var line in File.ReadAllLines(accountsPath))
    {
        var cols = line.Split(',');
        if (cols.Length < 7)
            continue;
        if (!int.TryParse(cols[1], out int id))
            continue;
        if (!double.TryParse(cols[6], NumberStyles.Float, CultureInfo.InvariantCulture, out double bal))
            bal = 0;
        string extra = cols.Length > 7 ? cols[7] : ("");

        Account acc = cols[0]
            switch
        {
```



Code Structure - Key Classes

The project is organized into four main components:

1. **Models**: like Person, UserAccount, and Account classes. They store data and define the structure.
2. **BankManager**: which contains all the business logic: authentication, account operations, and file handling.
3. **UI**: a static class that manages all console input and output, menus, colors, and tables.
4. **Program**: the entry point, connects the UI to the BankManager and controls the main menu loops. This separation of concerns makes the code clean, modular, and easy to maintain.



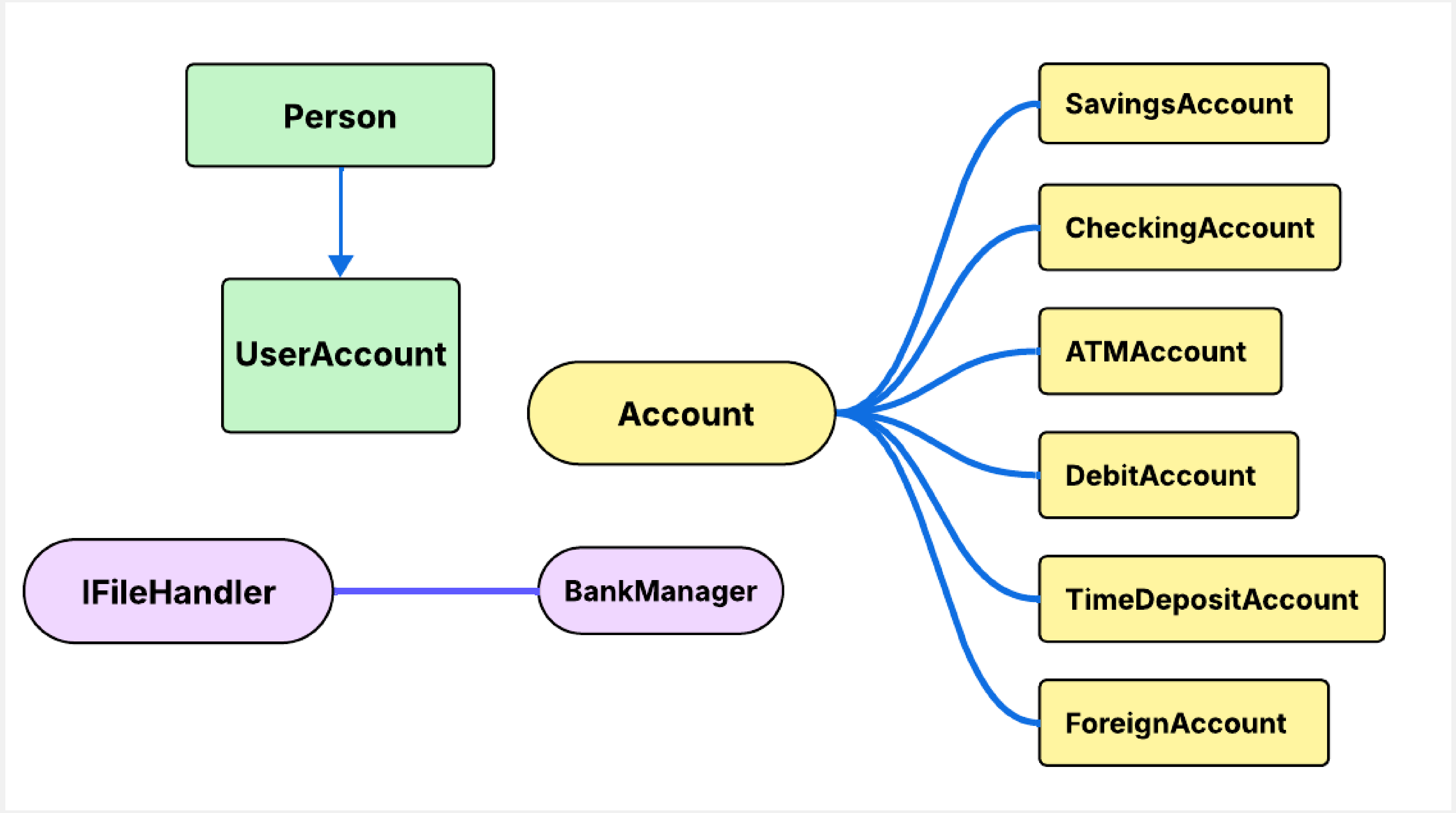
Code Structure - Code Walkthrough

Polymorphism & File Handling

- The abstract Account class defines a virtual ToCsv() method.
- Subclasses like SavingsAccount override this method to add their unique data (like InterestRate) to the CSV line.
- This allows one Save method to handle all 6 types.

Modularity & Reusability

- The static UI class is highly reusable. Methods like UI.PromptMenu() and UI.RenderAccountTable() are called from multiple places, reducing code duplication.



User Interface (UI)

Design & Usability

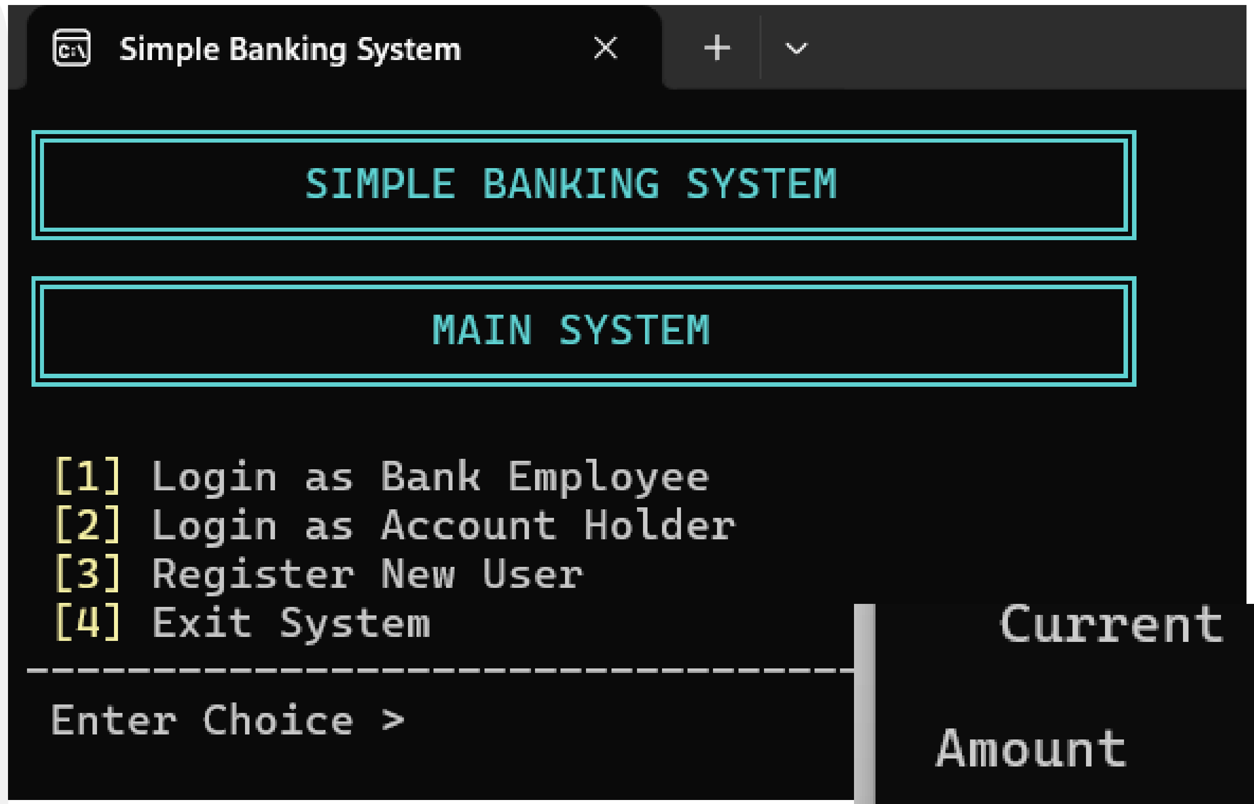
- Clean, menu-driven console interface
- Uses color coding (Cyan = titles, Red = errors)
- Includes UTF-8 box-drawing characters
- (╔, ═, ║) for polished menus & tables (RenderAccountTable)

Input / Output Handling

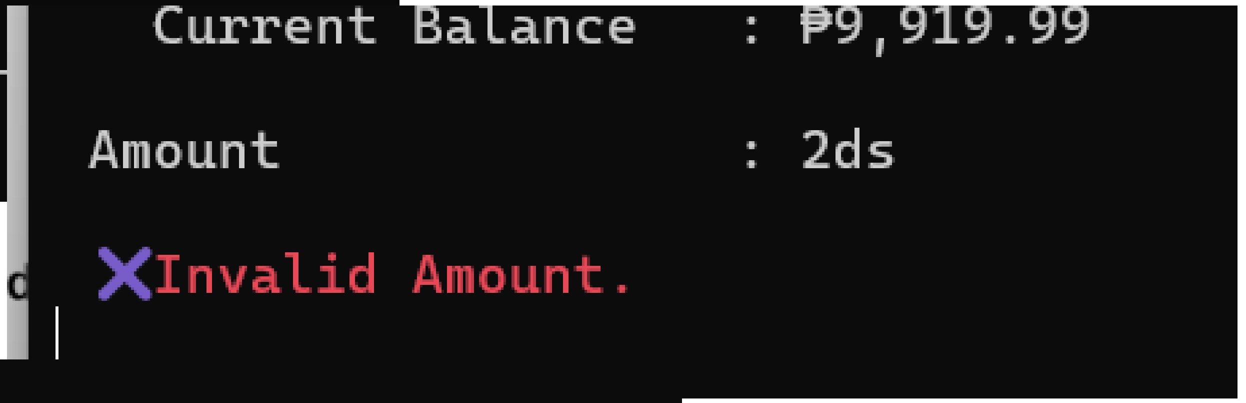
- **Input**: Menu choices (1, 2…), user info (strings), amounts (double)
- **Output**:
  - Auto-formatted menus
  - Account tables
  - Clear success & error messages

Error Handling

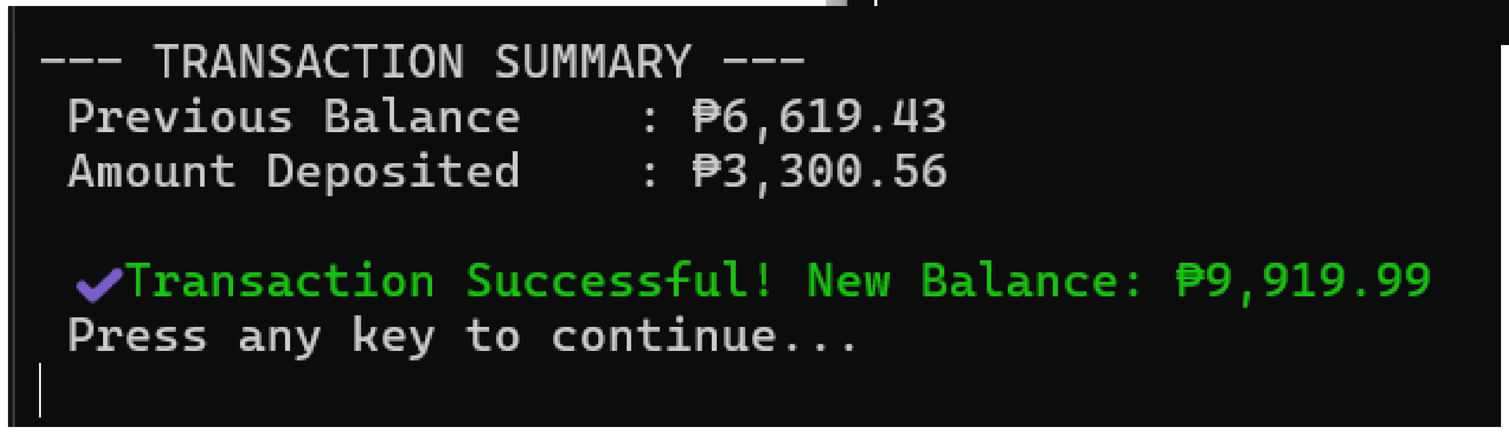
- **UI.PrintSuccess("✓ …")** for green success notifications
- **UI.PrintError("✕ …")** for red error alerts
- Validates input (**TryParse**) to prevent crashes and asks user to retry



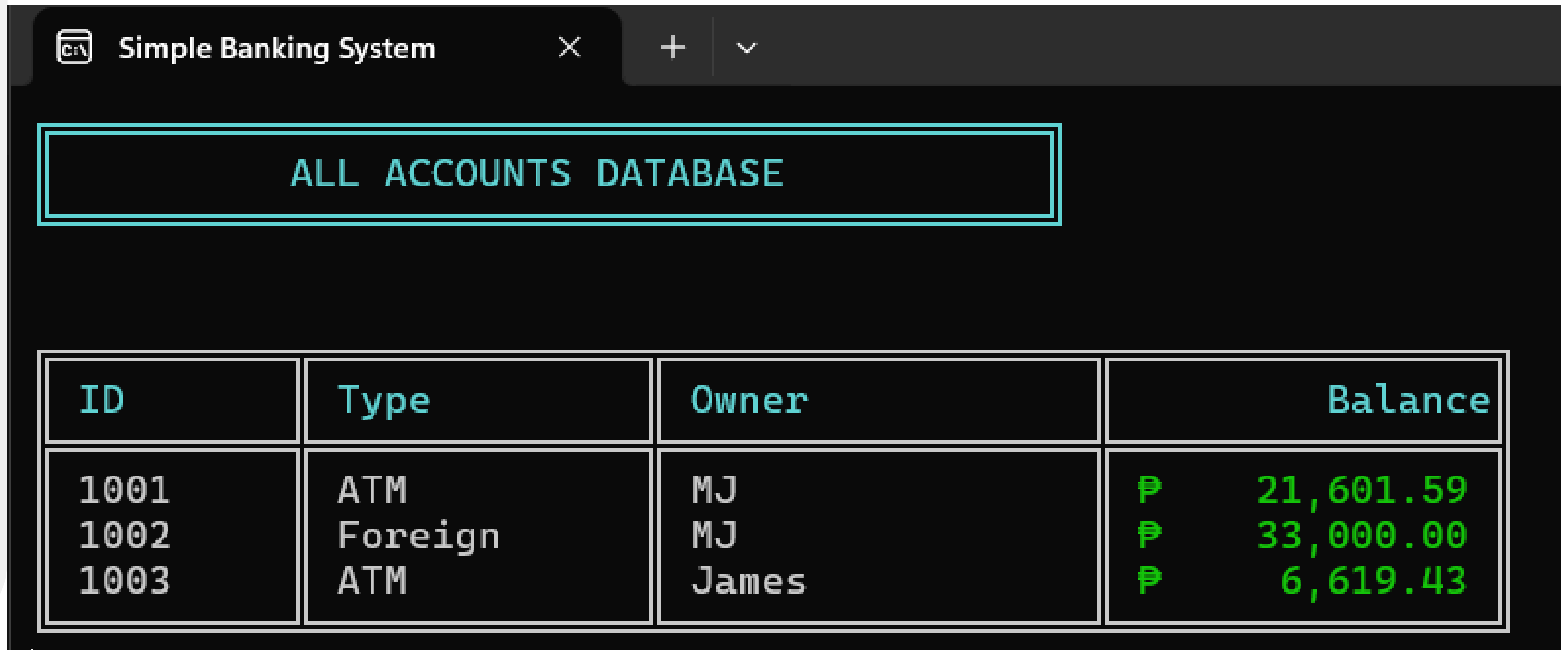
```
SIMPLE BANKING SYSTEM

MAIN SYSTEM

[1] Login as Bank Employee
[2] Login as Account Holder
[3] Register New User
[4] Exit System
--------------------------------
Enter Choice >
```



```
  Current Balance    : ₱9,919.99

Amount               : 2ds

✕Invalid Amount.
```



```
--- TRANSACTION SUMMARY ---
 Previous Balance     : ₱6,619.43
 Amount Deposited     : ₱3,300.56

 ✓Transaction Successful! New Balance: ₱9,919.99
 Press any key to continue...
```



```
ALL ACCOUNTS DATABASE

ID        Type         Owner              Balance
1001      ATM          MJ          ₱    21,601.59
1002      Foreign      MJ          ₱    33,000.00
1003      ATM          James       ₱     6,619.43
```

Challenges and Solutions

Storing Different Account Types

- **Problem**: Saving objects with different properties (e.g., InterestRate vs. Currency) into one file.
- **Solution**: Used a polymorphic ToCsv() method and a "type" column in the CSV to guide object reconstruction on load.

Secure Password Input

- **Problem**: Console.ReadLine() displays the password in plaintext.
- **Solution**: Implemented a Program.ReadPassword() helper function using Console.ReadKey(true) to mask input with *.

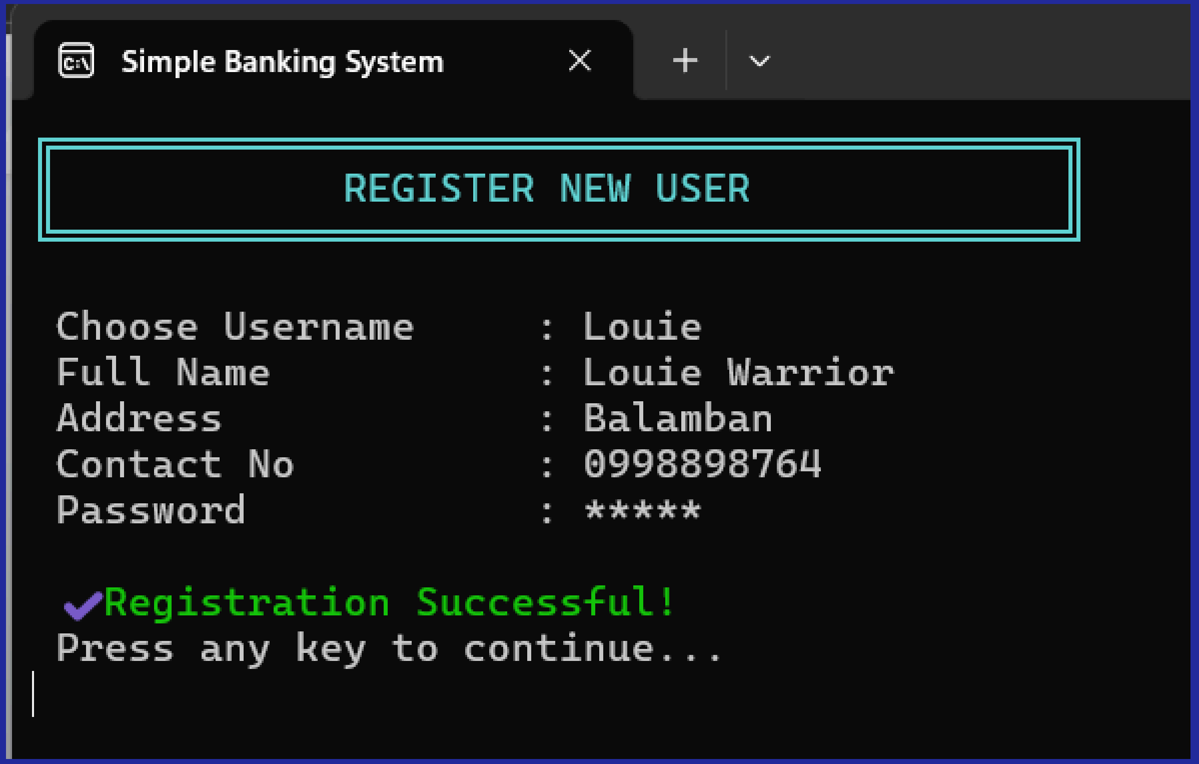
Code Clutter ("Spaghetti Code")

- **Problem**: Mixing Console.WriteLine logic inside the core BankManager business methods.
- **Solution**: Extracted all visual logic into the static UI class, ensuring the BankManager contains pure, testable business rules.

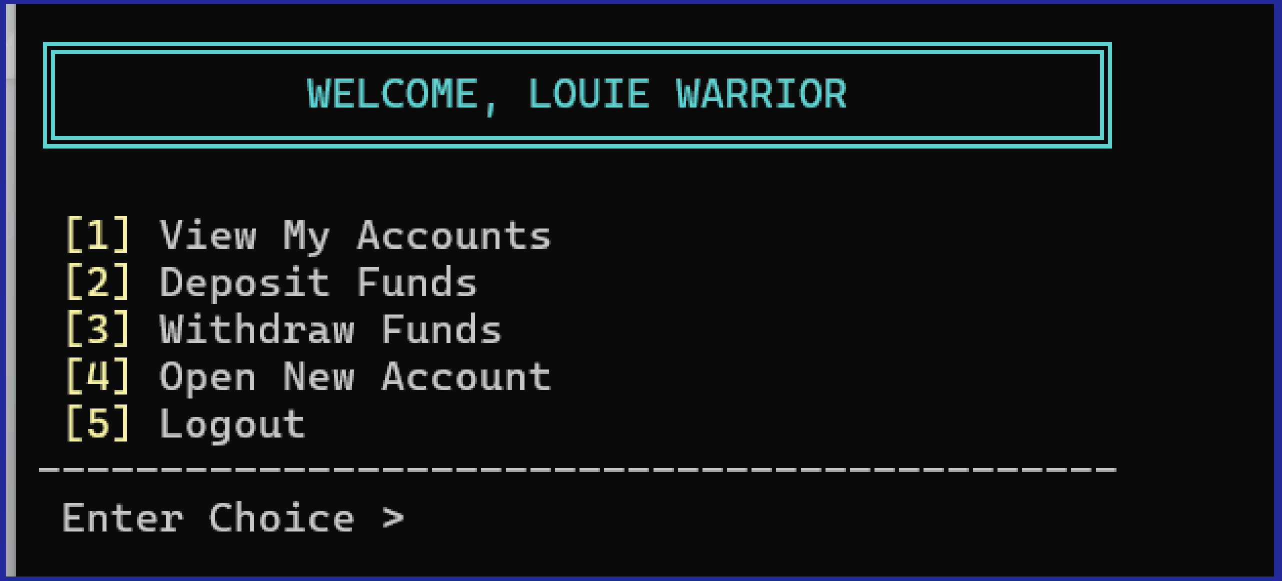
Testing

Test Cases :

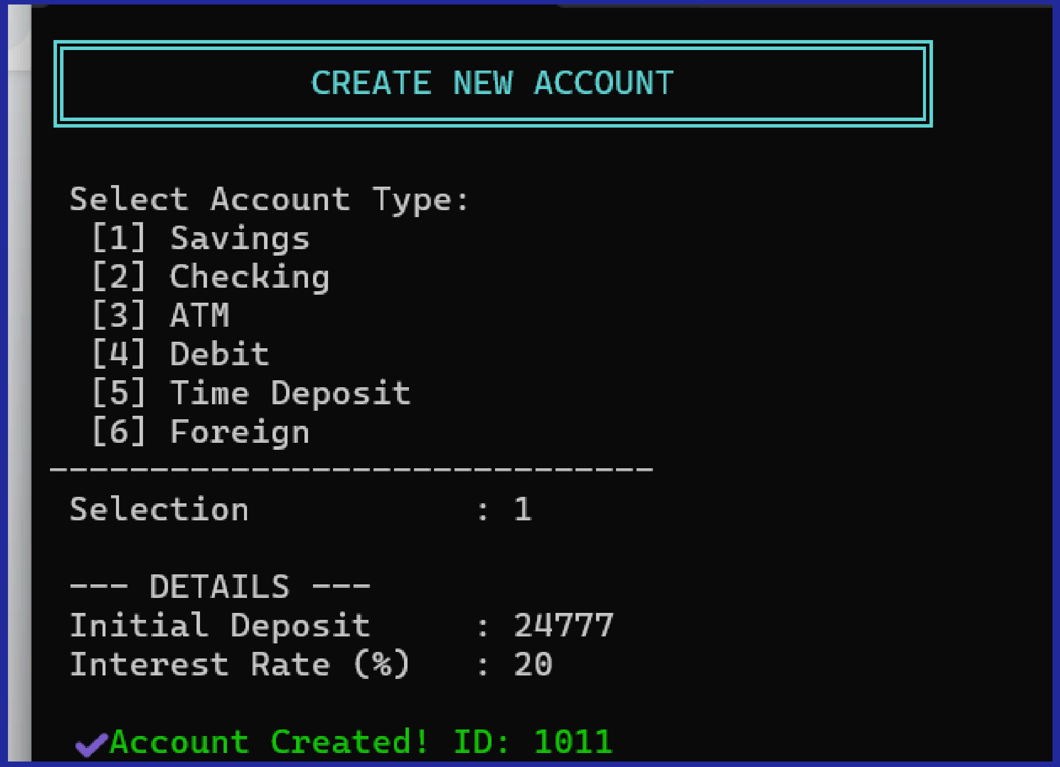
- Normal operations like registering, logging in, creating accounts, depositing, and withdrawing worked perfectly.



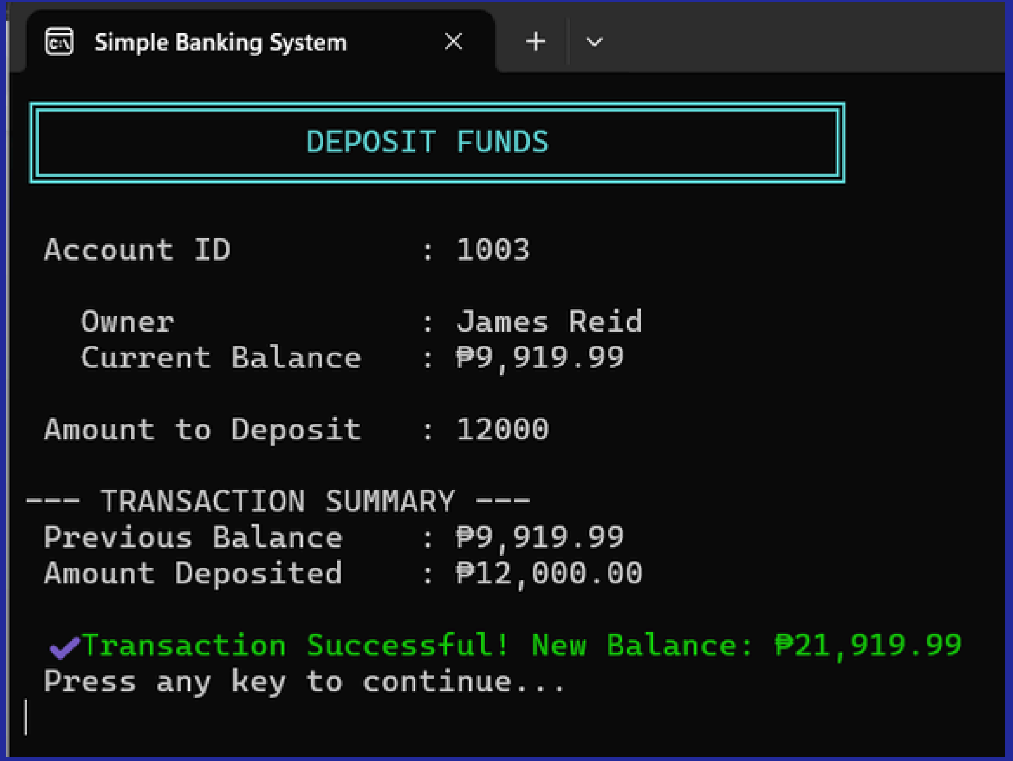
```
Simple Banking System

REGISTER NEW USER

Choose Username      : Louie
Full Name            : Louie Warrior
Address              : Balamban
Contact No           : 0998898764
Password             : *****

✔Registration Successful!
Press any key to continue...
```



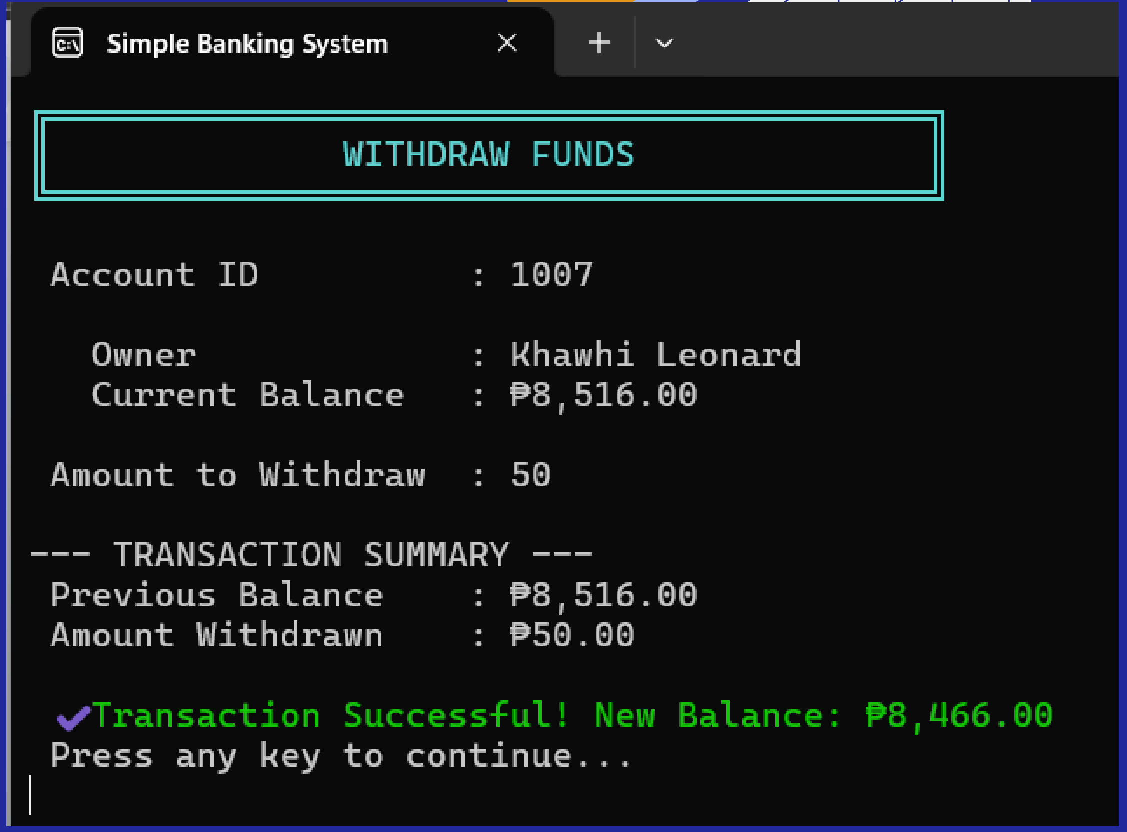
```
WELCOME, LOUIE WARRIOR

[1] View My Accounts
[2] Deposit Funds
[3] Withdraw Funds
[4] Open New Account
[5] Logout
---------------------------------------------
Enter Choice >
```



```
CREATE NEW ACCOUNT

Select Account Type:
 [1] Savings
 [2] Checking
 [3] ATM
 [4] Debit
 [5] Time Deposit
 [6] Foreign
------------------------------
Selection             : 1

--- DETAILS ---
Initial Deposit       : 24777
Interest Rate (%)     : 20

✔Account Created! ID: 1011
```



```
Simple Banking System

DEPOSIT FUNDS

Account ID           : 1003

  Owner              : James Reid
  Current Balance    : ₱9,919.99

Amount to Deposit    : 12000

--- TRANSACTION SUMMARY ---
Previous Balance     : ₱9,919.99
Amount Deposited     : ₱12,000.00

✔Transaction Successful! New Balance: ₱21,919.99
Press any key to continue...
```

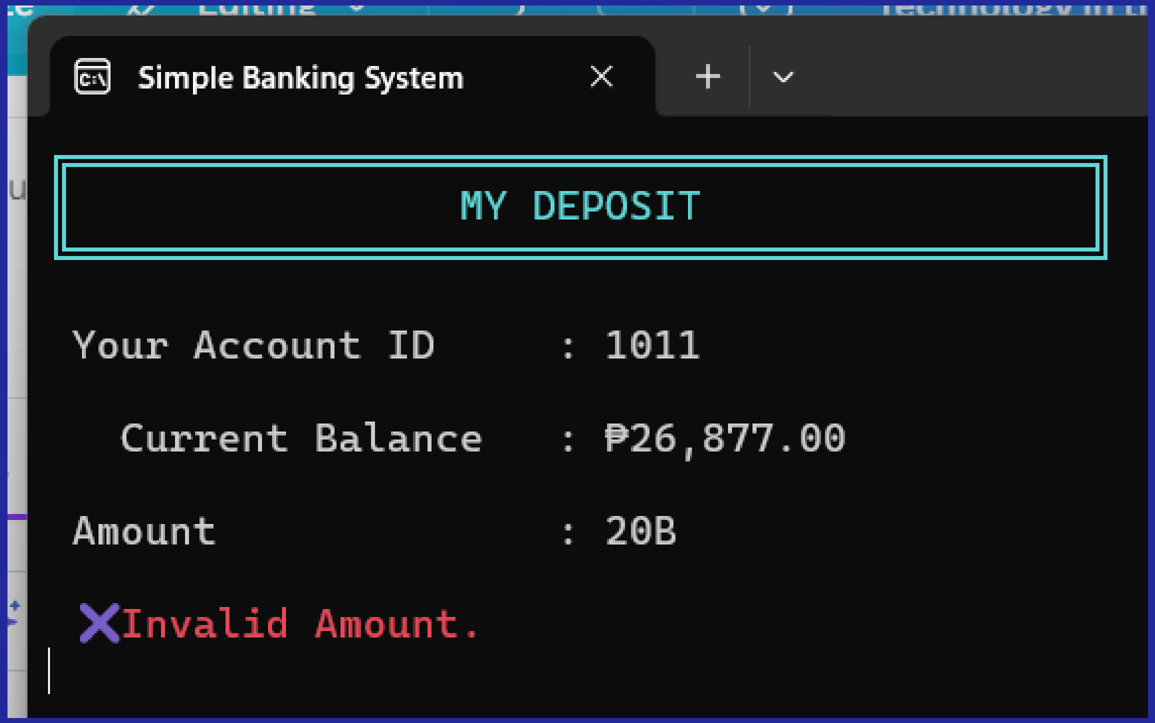


```
Simple Banking System

WITHDRAW FUNDS

Account ID           : 1007

  Owner              : Khawhi Leonard
  Current Balance    : ₱8,516.00

Amount to Withdraw   : 50

--- TRANSACTION SUMMARY ---
Previous Balance     : ₱8,516.00
Amount Withdrawn     : ₱50.00

✔Transaction Successful! New Balance: ₱8,466.00
Press any key to continue...
```

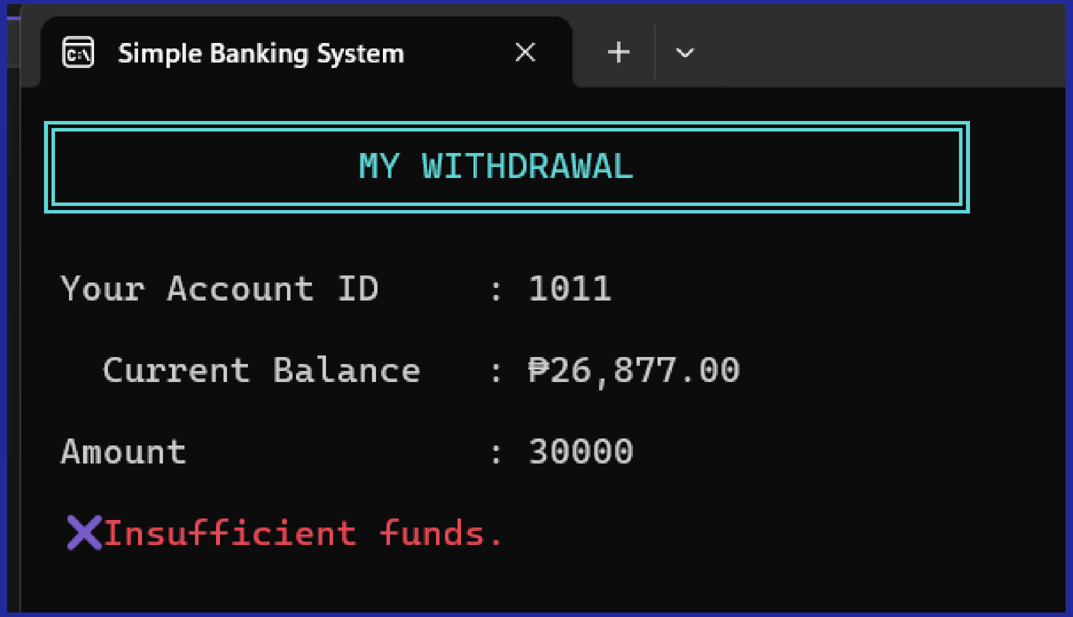
Testing

Test Cases (Edge Cases & Errors):

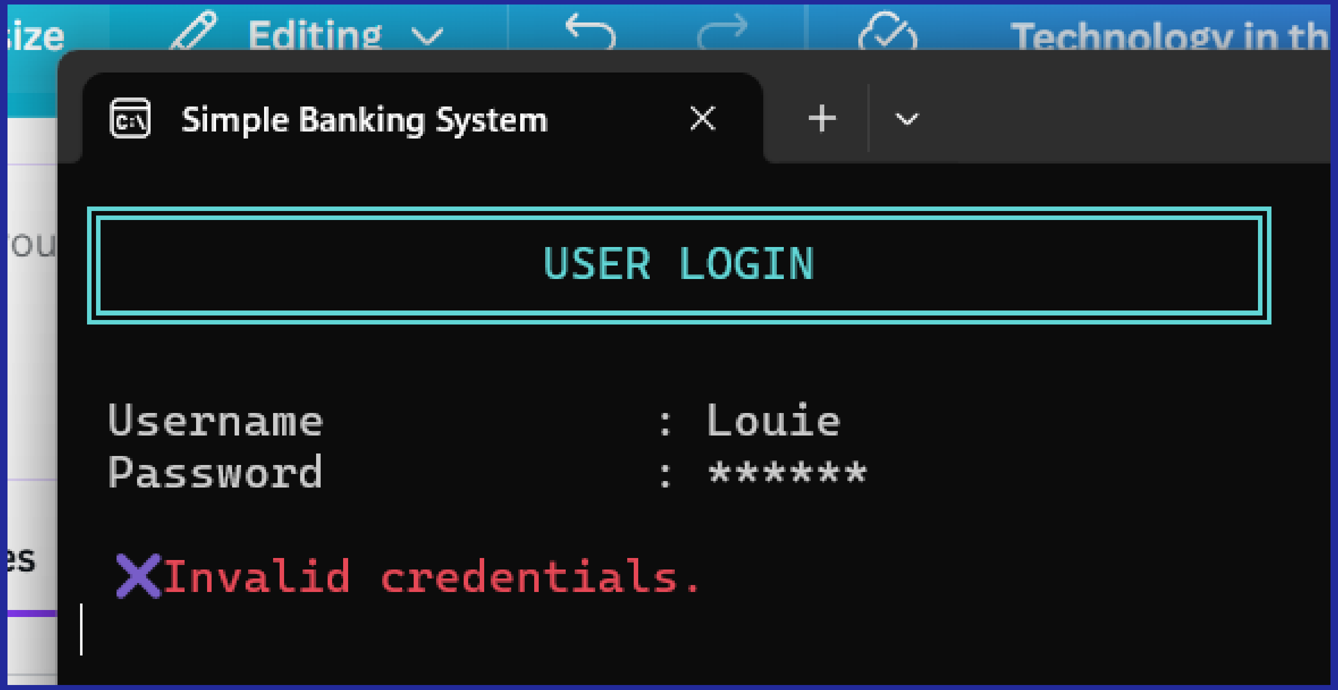
- Edge cases like withdrawing more than the balance, duplicate usernames, invalid numeric input, and incorrect login were all caught gracefully.



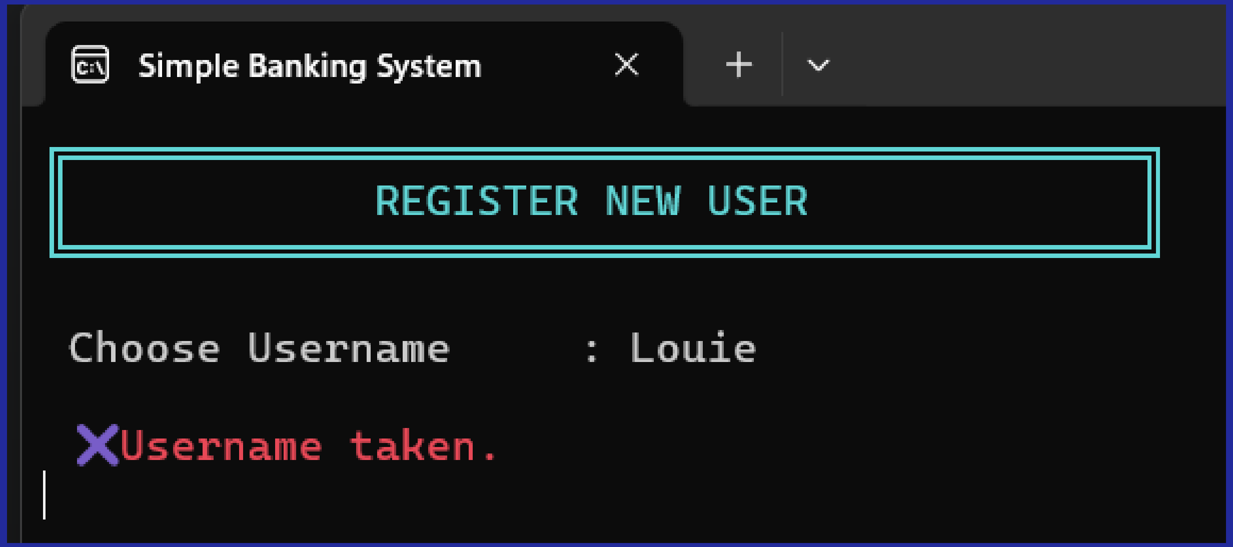
```
MY DEPOSIT

Your Account ID      : 1011
  Current Balance    : ₱26,877.00
Amount               : 20B
❌Invalid Amount.
```



```
MY WITHDRAWAL

Your Account ID      : 1011
  Current Balance    : ₱26,877.00
Amount               : 30000
❌Insufficient funds.
```



```
USER LOGIN

Username             : Louie
Password             : ******

❌Invalid credentials.
```



```
REGISTER NEW USER

Choose Username      : Louie

❌Username taken.
```

Testing

Results:

- All test cases passed successfully. Error handling routines caught all expected edge cases and invalid inputs.

Limitations:

- **Security**: Passwords are stored in plaintext (not hashed).
- **Scalability**: CSV files are not efficient for a large number of users.
- **Features**: No transaction history logging or account transfer.

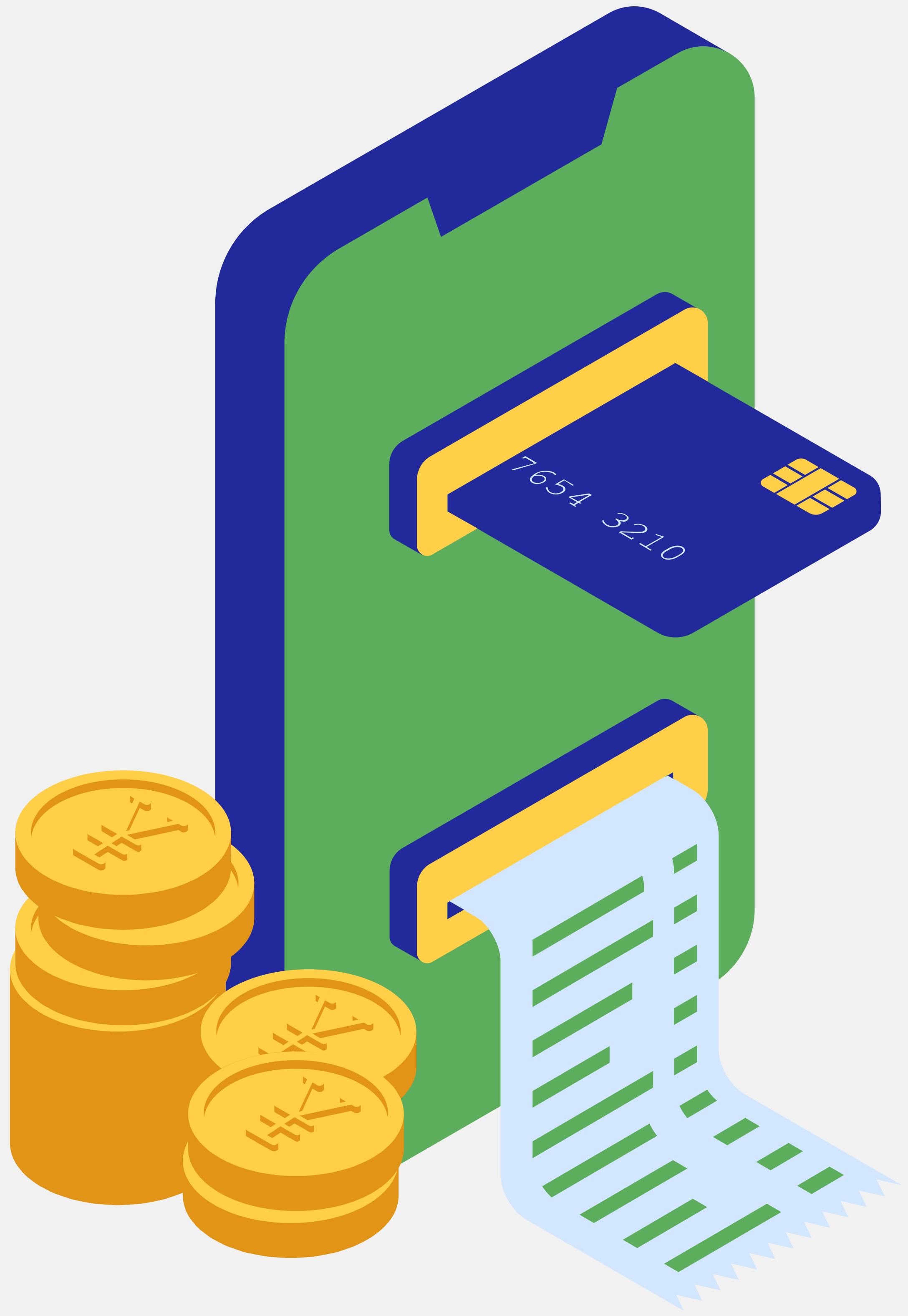
Future Enhancements

Planned Features:

- **Fund Transfers**: Allow a customer to transfer money between accounts.
- **Transaction History**: Log every deposit and withdrawal to a new file.
- **Interest Calculation:** Implement an automated calculation for SavingsAccount types.

Performance & Security Improvements:

- **Password Hashing:** Implement a real hashing algorithm (like BCrypt) for secure password storage.
- **Database Migration:** Replace the CSV file system with a relational database (like SQLite or SQL Server) using Entity Framework Core.
- **New UI:** Re-platform the logic into a new graphical application using WPF or .NET MAUI.



Conclusion

Reflection:

- This project successfully models a complex system in a simple console environment. It proves that with a strong architecture, even a console app can be powerful, maintainable, and user-friendly.

- **Key Takeaways (Skills Developed):**
    - **Advanced OOP in C#:** Practical application of Inheritance, Polymorphism, and Encapsulation.
    - **Data Persistence:** Robust file handling (System.IO) and data serialization to CSV.
    - **Clean Architecture:** The value of separating concerns (Model vs. Logic vs. UI).
    - **Reusable Components:** Building a modular UI library for future console projects.

THANK YOU!