Ahmed Farahat Sayed 42210249

Omar Ehab Mohamed 42210220

Mohamed Nasser Mohamed 42210159

Mostafa Hytham mymoon 42210060

Ziad Yasser Mohamed 42210162

Mohamed Khalid Kaseim 42210255

Banking System

# Abstract:

This project aims to create a simple banking system simulation using C++, providing users with the ability to create accounts, perform transactions, and manage different account types. The program features three main classes: `DataBase`, `Account`, and `Bank`, each serving distinct roles in the overall structure. The implementation employs file handling, inheritance, and dynamic memory allocation to achieve its functionality.

# Introduction:

The project revolves around a banking system simulation, where users can interact with a command-line interface to create accounts, view account details, and perform various transactions. The program is structured into classes to encapsulate different aspects of the banking system.

Classes Overview:

1. DataBase Class:

- The `DataBase` class handles file input/output operations for three account types: Diamond, Gold, and Silver.

- It establishes connections to corresponding data files using ifstream and manages their closure in the class destructor.

2. Account Class:

- The `Account` class represents a generic bank account with essential attributes such as account number, account holder name, and balance.

- It provides methods for depositing, withdrawing, and displaying account information.

- A derived class, `CheckingAccount`, extends the functionality by introducing an overdraft limit for withdrawals.

3. Bank Class:

- The `Bank` class manages an array of `Account` pointers, providing essential banking operations.

- Users can create accounts, view account details, and perform transactions like deposit, withdrawal, cash out, coin transfer, and inter-account transfer.

- The class dynamically manages memory for accounts and adjusts capacity as needed.

Implementation Details:

1. File Handling:

- The `LinkDataBase` method in the `DataBase` class establishes connections to data files for Diamond, Gold, and Silver accounts.

- Error handling ensures that the program exits gracefully if file operations encounter issues.

2. Account Types:

- Although the program currently treats all account types similarly, the architecture allows for future expansion to incorporate unique behaviors for Diamond, Gold, and Silver accounts.

3. Memory Management:

- The `Bank` class handles dynamic memory allocation for accounts, ensuring proper memory management.

- A resizing mechanism adjusts the array's capacity to accommodate a growing number of accounts.

4. User Interface:

- The program offers a user-friendly interface with a main menu and an operation menu, allowing users to navigate through various banking functionalities.

In this section, users are presented with a main menu that includes options for creating an account, viewing account details, making transactions, and exiting the program. The use of a `do-while` loop ensures that the program continues to execute until the user chooses to exit.

Create Account (Option 1):

- Users can create a new account by selecting option 1. The `createAccount` method in the `Bank` class guides users through the account creation process. They are prompted to enter details such as account number, account holder name, initial balance, and account type (Diamond, Gold, Silver, or a CheckingAccount).

- The program then dynamically manages memory to store the new account and ensures that the account data is saved to the corresponding data file.

View Account Details (Option 2):

- Option 2 allows users to view the details of an existing account. After entering the account number, the program uses the `viewAccountDetails` method to display information such as account number, account holder name, and balance.

Make Transaction (Option 3):

- Option 3 initiates the transaction process. Users are prompted to enter their account number, and the program then calls the `makeTransaction` method in the `Bank` class. This method provides a menu of transaction options, such as deposit, withdrawal, cash out, coin transfer, and inter-account transfer. Users can perform these transactions based on their preferences.

Exit Program (Option 4):

- If the user chooses option 4, the program displays a closing message and exits gracefully.

This organized and user-friendly interface makes the banking simulation accessible to users, enabling them to perform various tasks efficiently.

Conclusion:

In conclusion, this C++ banking system simulation project successfully encapsulates essential banking operations within a modular and object-oriented framework. The design allows for flexibility and potential expansions, making it a foundation for further development and refinement. By addressing recommended improvements, the program can become more robust, providing a reliable and feature-rich simulation of a banking system. The current implementation already serves as an educational tool for understanding file handling, inheritance, dynamic memory allocation, and user interface design in C++ programming.