Bank Account Management System Documentation

I, Object-Oriented Analysis (OOA)

The OOA analysis follows a four-step model to identify objects, attributes, methods, and inheritance relationships for the bank account management system.

-Step 1: Identify Objects (Nouns): The main entities include:
Account: Represents a basic bank account.
SavingsAccount: A specialized account inheriting from Account.
Customer: Represents a bank customer owning accounts.
Transaction: Represents a financial transaction (deposit, withdrawal, trans- fer).

-Step 2: Identify Attributes (Descriptive Nouns):
Account: accountNumber (string), balance (double), ownerName (string), transactionHistory (vector<Transaction>).
SavingsAccount: Inherits from Account, adds interestRate (double).
Customer: name (string), id (string), accounts (vector<Account*>).
Transaction: amount (double), type (string).

-Step 3: Identify Methods (Verbs):
Account: deposit(), withdraw(), debit() (virtual), credit(), displayInfo(), op- erators +=, ==.
SavingsAccount: Overrides debit(), adds applyInterest().
Customer: openRegularAccount(), openSavingsAccount(), getTotalBalance(), displayInfo().
Transaction: getAmount(), getType().

-Step 4: Identify Inheritance Relationships:
SavingsAccount publicly inherits from Account, overriding debit() to include withdrawal fees and adding applyInterest().

II, Class Design Explanation

The class design leverages OOP principles to model the banking system:

- Encapsulation: Attributes of Account, SavingsAccount, Customer, and Transaction are private, with public methods for access and manipulation.

- Inheritance: SavingsAccount inherits from Account, reusing attributes like balance and methods like deposit(). The debit() method is overridden to apply a $2 with- drawal fee.

- Operator Overloading:

+ The += operator in Account adds a Transaction to the history and updates the balance.
+ The == operator compares two accounts based on accountNumber.

- Polymorphism: The debit() method is virtual in Account, allowing SavingsAccount to override it with fee logic.

Inheritance and operator overloading make the code intuitive (e.g., account += Trans- action(50.0, ”bonus”)) and extensible.

III, Code Walkthrough

Below are key excerpts from the C++ code:

```
// Lớp cơ sở Account quản lý tài khoản ngân hàng
class Account {
protected:
    string accountNumber;       // Số tài khoản
    double balance;             // Số dư
    string ownerName;           // Tên chủ tài khoản
    vector<Transaction> transactionHistory; // Lịch sử giao dịch

    // Thêm giao dịch với số tiền có dấu
    void addSignedAmount(double signedAmt, const string& type) {
        balance += signedAmt;
        transactionHistory.push_back(Transaction(signedAmt, type));
    }

public:
    Account(const string& accNum, const string& ownName, double initBal = 0.0)
        : accountNumber(accNum), balance(initBal), ownerName(ownName) {}
    virtual ~Account() {}
```

```cpp
// Nạp tiền vào tài khoản
void deposit(double amount) {
    if (amount > 0) {
        addSignedAmount(amount, "deposit");
    } else {
        cout << "Deposit amount must be positive." << endl;
    }
}

// Rút tiền hoặc chuyển khoản (ảo để lớp con ghi đè)
virtual bool debit(double amount, const string& type) {
    if (amount <= 0) {
        cout << "Debit amount must be positive." << endl;
        return false;
    }
    if (balance < amount) {
        cout << "Insufficient funds for debit." << endl;
        return false;
    }
    addSignedAmount(-amount, type);
    return true;
}
```

- **Inheritance**: SavingsAccount overrides debit() to apply a $2 fee for withdrawals, ensuring sufficient balance for both amount and fee.

IV,Test Results

The program was tested in the main() function with the following scenarios:
- Create a customer (John Doe, ID C001).
- Open a regular account (A001, $1000) and a savings account (S001, $5000, 5% interest).
- Perform transactions: Deposit $200 to A001, deposit $300 and withdraw $100 (with $2 fee) from S001, transfer $300 from A001 to S001.
- Apply interest to S001.
- Use += to add a $50 bonus transaction to A001.
- Compare accounts A001 and A002 (result: not equal).

- Sample Output:
Accounts A001 and A002 are not equal.
Customer Name: TaiHuynh
Customer ID: C001
Total Balance: $6907.9
Owned Accounts:
Account Number: A001
Owner: TaiHuynh
Balance: $450
Transaction History:
deposit: $200
withdrawal: $-500
transfer: $-300
bonus: $50

Account Number: S001
Owner: TaiHuynh
Balance: $5757.9
Transaction History:
deposit: $300
withdrawal: $-100
fee: $-2
interest: $259.9
transfer: $300

Account Number: A002
Owner:TaiHuynh
Balance: $700
Transaction History:

Explanation: The diagram illustrates the process of opening an account and performing a deposit, from the user calling Customer to Account creating a Transaction

V, LLM Usage

- > I used an LLM to assist in development. Specific prompts in- cluded:
”Suggest ways to overload operators for a bank account class.”
”Explain how inheritance can be used in a bank account management system.”

The LLM suggested overloading += for adding transactions and == for comparing ac- counts, and explained how SavingsAccount could inherit from Account. I implemented the code independently based on these ideas.