BANK ACCOUNT MANAGEMENT SYSTEM – REPORT

Name: Huỳnh Vũ Minh Hiền **Student's ID:** 24110091
Subject: Object-Oriented Programming

I. Object-Oriented Analysis (OOA) Model Account

1. Objects and Attributes

- SavingsAccount (specialized Account)

- Customer

- Transaction

Attributes

- Account: accountNumber, ownerName, balance, history

- SavingsAccount (inherits Account): interestRate, withdrawalLimit

- Customer: name, customerID, accounts

- Transaction: amount, type, date

2. Identify Methods

- Account: deposit(), withdraw(), displayInfo(), getBalance(), operator+= (add transaction), operator== (compare accounts)

- SavingsAccount: withdraw() (overridden with limits), applyInterest(), displayInfo() (overridden)

- Customer: openAccount(), viewAccounts(), totalBalance(), displayInfo()

- Transaction: displayInfo(), getType(), getAmount()

3. Inheritance Relationships

- Account is a base class.

- SavingsAccount inherits from Account.

- Customer contains multiple accounts (composition).

- Transaction objects are associated with Account.

II. Overview of the Bank Account Management System

- Account (Base Class): Represents a general bank account with account number, owner name, balance, and transaction history. Provides deposit or withdraw operations.

- SavingsAccount (Derived Class): Extends Account, adds attributes (interestRate, withdrawalLimit) and overrides withdraw() to enforce rules.

- Transaction: Represents each operation. Used to update history and track account activities.

- Customer: Represents a client, storing personal details and list of owned accounts. Provides methods to open new accounts and check total balance.

III. Code Walkthrough

- Transaction class: Stores and displays transaction details (type, amount, date).

- Account class: Base class with balance management, transaction history, and operator overloading (+= to add transaction, == to compare accounts).

- SavingsAccount class: Derived from Account. Overrides withdraw() and adds interest application.

- Customer class: Manages multiple accounts, can view all accounts and calculate total balance.

- Main function: Creates accounts and customers, performs deposits or withdrawals, applies interest, books transactions, and demonstrates operator overloading and polymorphism.

IV. System Operations

- Account: Create accounts, deposit or withdraw money, display details, compare with ==, add transactions using +=.

- SavingsAccount: can apply interest to increase balance.

- Customer: Create profiles, open accounts, display all accounts, calculate total balance.

- Transaction: Create deposit or withdraw or transfer records, display transaction details.

V. Testing the System & Results

Test Case 1: Create and Display Accounts

```
Account acc1(101, "Truong", 500);
```

SavingsAccount acc2(102, "Truong", 1000, 0.05);

Account acc3(201, "Hien", 700);

=> Output: Correctly displays account number, owner name, and balance.

Test Case 2: Deposit and Withdraw (Account)

acc1.deposit(200, "2025-09-22");

acc1.withdraw(100, "2025-09-22");

=> Output: Balance updated correctly (600).

Test Case 3: Savings Account with Rules and Interest

acc2.deposit(500, "2025-09-22");

acc2.withdraw(1500, "2025-09-22");

acc2.withdraw(500, "2025-09-22");

acc2.applyInterest("2025-09-22");

=> Output: Shows error message when withdrawing more than balance, successful withdrawal within the limit, and balance increased by 5% interest.

Test Case 4: Operator Overloading (+=)

Transaction t1("Deposit", 300, "2025-09-22");

acc3 = acc3 += t1;

=> Output: acc3's balance increases to 1000 and the transaction is recorded in history.

Test Case 5: Account Comparison (==)

if (acc1 == acc3) cout << "Have the same balance";

else cout << "Don't have the same balance";

=> Output: Correct comparison based on balance. (acc1 = 600, acc3 = 1000 → "Don't have the same balance").

VI. LLM Usage

I used ChatGPT (LLM) for:

- I relied on AI to suggest additional attributes and improve the completeness of my work.

- I asked it to explain the usage of transactions.

- I requested debugging hints for errors in operator overloading.

- I asked it to turn my ideas into code.

- Based on those suggestions, I wrote my own code independently.

VII. Conclusion

This project demonstrates Object-Oriented Programming concepts:

- Encapsulation: Private attributes with methods for access.

- Inheritance: SavingsAccount extends Account.

- Polymorphism: Overridden withdraw method.

- Operator Overloading: += for adding transactions, == for comparing accounts.

- Composition: Customer contains multiple accounts, and Account contains a history of transactions.