Object - Oriented Programming

Name: Lê Sĩ Nhật Khuê **Student's ID:** 24110102

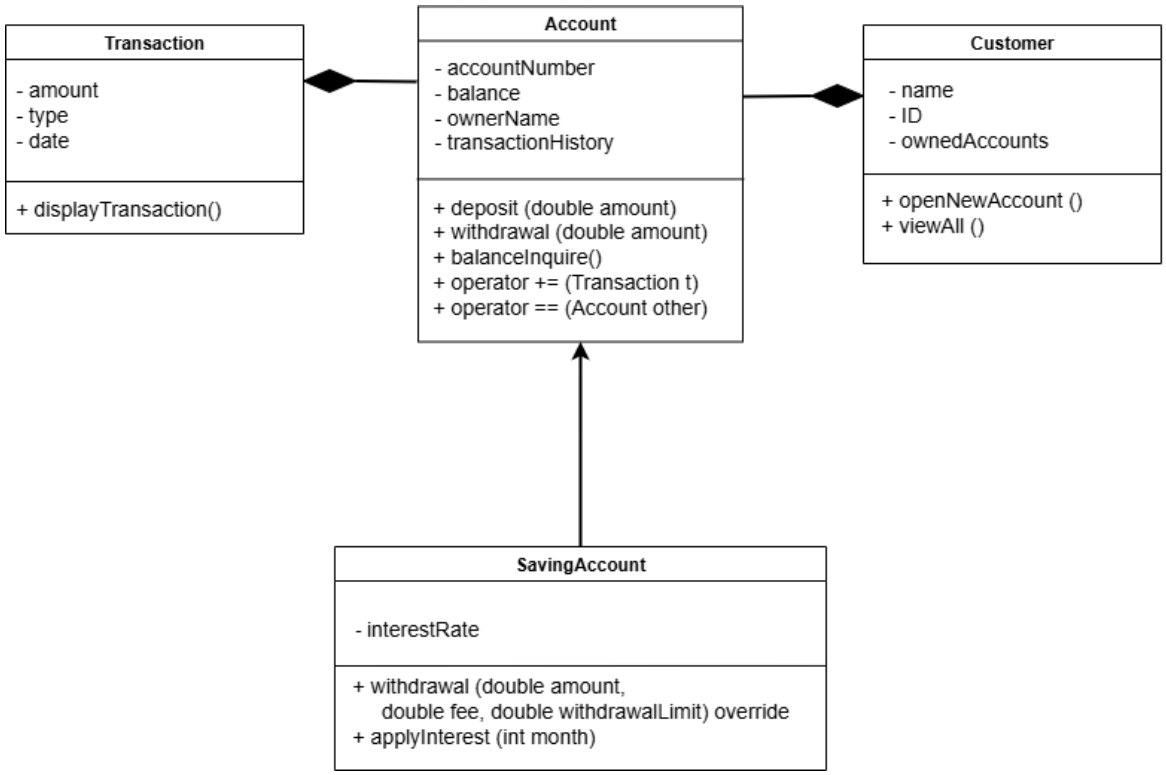
Assignment 6 – Week 5: Bank Account Management System

I/ Object-Oriented Analysis (OOA) Model

Base on the information provided, we have some analysis

- About **objects**, we can define some objects such as Account, SavingAccount, Customer, Transaction.
- About **attributes** for each object:
    + For **Transaction:** amount, type, date.
    + For **Account:** accountNumber, balance, ownerName, transactionHistory (to store list of transaction history).
    + For **SavingAccount:** inherits from Account and add interestRate.
    + For **Customer:** name, ID, ownedAccounts.
- About **methods** for each object:
    + For **Transaction:** displayInfo.
    + For **Account:** deposit, withdrawal, balanceInquire, displayInfo and two operator overloading "+=", "==".
    + For **SavingAccount:** inherits from Account, override withdrawal and add applyInterest method.
    + For **Customer:** openNewAccount, displayAccounts, getTotalBalance.
- About **inheritance relationships**:
    + SavingAccount inherits from Account.
    + One account can have many transactions.
    + One customer can own multiple accounts (both normal and saving).

II/ Overview the Bank Account Management System



The UML class diagrams

The system models a simplified banking environment that supports customers, accounts, transactions, and savings features. The design follows object-oriented principles such as encapsulation, inheritance, and operator overloading.

Class Design Explanation:

- The **Transaction class** stores essential information about a single transaction, including the amount, type (*deposit, withdrawal, transfer*), and date. It also provides a method to display transaction details.

- The **Account class** represents a basic bank account with attributes like account number, owner name, balance, and transaction history. It provides methods for deposit, withdrawal, balance inquiry, and displaying account details.

- The **SavingAccount class** extends Account using inheritance, adding an interestRate attribute and an applyInterest method. It also overrides the withdrawal method to enforce specific rules such as fees and minimum balance requirements. This demonstrates polymorphism, as the behavior of withdrawal depends on the actual account type.

- The **Customer class** manages customer information (name, ID) and their list of accounts. It supports operations such as opening new accounts, displaying all accounts, and calculating total balance.

Use of Inheritance:

Inheritance is applied between Account (base class) and SavingAccount (derived class). This relationship reflects the real-world hierarchy where a savings account is a specialized type of account. It allows shared attributes and methods (e.g., balance, deposit) to be reused, while adding specific features (interest rate) only to savings accounts.

Use of Operator Overloading:

Two operators are overloaded in the Account class:

- "**operator +=**" is used to add a transaction directly to an account's transaction history. This provides a natural and concise syntax when recording new transactions.

- "**operator ==**" compares two accounts by their balance, enabling intuitive equality checks between accounts.

In addition to the main classes, the system uses a few supporting functions and structures:

- struct Date: Represents a simple date with day, month, and year. It is used by the Transaction class to record when a transaction occurs.

- getTypeTransaction: A utility function that maps an integer transaction type (1 is deposit, 2 is withdrawal and 3 is transfer) into a descriptive string. This ensures consistency when displaying transaction information.

- getToday: A function that leverages the <ctime> library to obtain the current system date and return it as a Date struct. It simplifies the process of assigning accurate dates to transactions automatically.

These helper functions improve code readability and reduce duplication, allowing classes such as Transaction and Account to focus on their core responsibilities while reusing shared functionality.

III/ How to do this assignment

To complete this assignment, I first reviewed the concepts of object-oriented programming that I had learned in class, such as encapsulation, inheritance, and operator overloading. Based on these principles, I identified the main classes required for a simple banking management system: Account, SavingAccount, Transaction, and Customer.

I then started by designing the structure of each class with its attributes and methods. During the coding process, I implemented operator overloading (+= and ==) to practice polymorphism and make the interaction with transactions more intuitive. I also used inheritance between Account and SavingAccount to avoid code duplication and to demonstrate how overriding works.

In addition to my own understanding, I also used **ChatGPT** as a supportive tool. It helped me clarify some programming concepts, especially with date handling using <ctime>, writing menu-driven programs, and explaining how to implement operator overloading correctly. The AI support allowed me to save time and focus more on understanding the logic rather than just syntax details.

However, there were some difficulties I encountered:

- At first, I struggled with designing the relationships between classes (especially how Customer should manage multiple Account objects).

- Another challenge was ensuring that the menu system worked interactively without skipping inputs. I had to learn how to handle user input carefully.

- Implementing operator overloading also required me to review C++ syntax multiple times, as it was a new concept for me.

Overall, this assignment was a good opportunity to combine classroom knowledge with practical problem-solving. Although I faced challenges, with the support of ChatGPT and the materials learned in class, I was able to complete the system successfully.