Report: Bank Account Management System

Name: Nguyễn Huỳnh Minh Tuyết
Student ID: 24110144

---

1. OOA Analysis (4-step model)

Step 1: Identify objects (nouns)

- Transaction
- Account
- SavingsAccount
- Customer
- Operations (system controller)

Step 2: Identify attributes (descriptive nouns)

- **Transaction**: amount, type (deposit/withdraw), date
- **Account**: accountNumber, accountName, balance, transactions
- **SavingsAccount**: inherits Account + interestRate, withdrawalLimit, withdrawalFee
- **Customer**: name, ID, accounts (list)
- **Operations**: list of accounts, list of customers, list of transactions

Step 3: Identify methods (verbs)

- **Transaction**: getters/setters, operator<< for printing
- **Account**: deposit(), withdraw(), operator+= (add transaction), operator== (compare balances)
- **SavingsAccount**: applyInterest(), override withdraw()
- **Customer**: addAccount(), getTotalBalance()
- **Operations**: addAccount(), addCustomer(), addTransaction(), performTransaction(), applyInterestToSavings(), displayAccountInfo()

Step 4: Inheritance relationships

- **SavingsAccount : Account** (single inheritance).
  SavingsAccount extends Account to add interest and withdrawal policies.

---

2. Class Design & Why Inheritance is Used

Design overview:

- **Transaction** records details of deposits/withdrawals.
- **Account** represents a general bank account, storing balance and transactions.
- **SavingsAccount** inherits from Account, adds interest rate and withdrawal rules.
- **Customer** holds customer data and list of accounts.
- **Operations** manages customers, accounts, and transactions.

Why inheritance is used:

- **Reuse**: SavingsAccount reuses all attributes/methods of Account.
- **Specialization**: SavingsAccount overrides withdraw() and adds applyInterest().

---

3. Code Walkthrough

- **Transaction**:
  - Attributes: amount, type, date.
  - Operator << prints a formatted transaction (e.g., [2024-10-01] deposit: 2000).
- **Account**:
  - deposit() and withdraw() adjust balance.
  - Operator += allows adding a Transaction: applies it (deposit/withdraw) and records it.
  - Operator == compares balances.
- **SavingsAccount**:
  - Overrides withdraw() to enforce withdrawal limit and fee.
  - applyInterest() deposits interest into account.
- **Customer**:
  - Can add accounts, get total balance across them.

- **Operations**:
  - performTransaction() finds account and applies a Transaction using operator+=.
  - applyInterestToSavings() applies interest to all savings accounts.
  - displayAccountInfo() shows account details and transaction history.

---

4. Test Results

Sample console output:

Account Number: A001
Account Name: Nguyen Van A - Savings
Balance: 9190
Transactions:
[today] create: 10000
[2024-10-01] deposit: 2000
[2024-10-03] withdraw: 3000
[today] interest: 190

Account Number: A002
Account Name: Nguyen Van A - Checking
Balance: 4000
Transactions:
[today] create: 5000
[2024-10-02] withdraw: 1000

Explanation:

- Customer and accounts were added successfully.
- Transactions were applied correctly: deposits increased balance, withdrawals decreased balance.
- SavingsAccount applied interest (5% of balance after transactions).
- Transaction history was displayed using operator<<.

---

5. LLM Usage (ChatGPT & Copilot)

During development, I used **ChatGPT** and **Copilot** to refine my design and fix mistakes.

How the LLM helped:

- **Transaction class**:
  - ChatGPT suggested moving Transaction above Account so that Account could include a vector<Transaction> cleanly.
  - ChatGPT also explained clearly how to implement and use operator<< for Transaction printing.
- **Operators in Account**:
  - ChatGPT helped me understand how to properly implement operator+= for adding transactions and operator== for comparing balances.
  - This made my code shorter, easier to read, and less error-prone.
- **Code clarity**:
  - ChatGPT helped rewrite some verbose code into cleaner functions.
  - It also pointed out errors (e.g., missing initialization, operator placement).
- **Testing (main function)**:
  - Copilot suggested example customers, accounts, and transactions to avoid mistakes when preparing test data.
  - Thanks to these suggestions, the output was consistent and demonstrated all key features (deposit, withdraw, interest, printing).

Example prompts (paraphrased):

- "How to implement operator+= in a bank account class to add transactions?"
- "Why should the Transaction class be declared before Account?"
- "Explain how operator<< works for printing objects in C++."

Notes:

- I used **ChatGPT** mainly for refactoring, fixing operator overloads, and clarifying concepts.
- I used **Copilot** for inline suggestions of test data.
- All final code was written and verified by me.