1. Object-Oriented Analysis (OOA)

- **Main Objects**:
  - **Transaction**: represents a banking transaction (deposit, withdraw, fee, interest).
  - **Account**: basic bank account (account number, balance, owner, transaction history).
  - **SavingAccount**: special type of account, inherits from Account, adds interest rate and withdrawal fee.
  - **Customer**: represents a bank customer who owns multiple accounts.
- **Functional requirements**:
  - Deposit, withdraw, add interest.
  - Record transaction history.
  - Compare transactions (operator==).
  - Use operator += to add transactions to accounts.

---

2. Class Design

- **Inheritance**:
  - SavingAccount inherits from Account to reuse account management code.
  - Adds **interest rate** and **withdrawal fee** features.
- **Operator Overloading**:
  - operator<<: print account/transaction details.
  - operator==: compare two transactions (same amount, type, and date).
  - operator+=: add a transaction to an account, update balance, and log the transaction.

👉 Reason: it makes the code cleaner and more natural. For example:

*acc1 += t1;

instead of calling separate methods like deposit() or withdraw().

---

3. Code Description

- **Transaction**:
    - Attributes: amount, type, date.
    - Methods: display, getters, operator<<, operator==.
- **Account**:
    - Attributes: accountnumber, balance, ownername, history.
    - Methods: deposit, withdraw, displayinfo, operator+=.
- **SavingAccount**:
    - Adds interestRate.
    - Overrides withdraw() (with a 1% fee).
    - Adds addinteresrate() method.
- **Customer**:
    - Stores multiple accounts (vector<Account*>).
    - Calculates total balance and displays account info.

---

4. Test Results

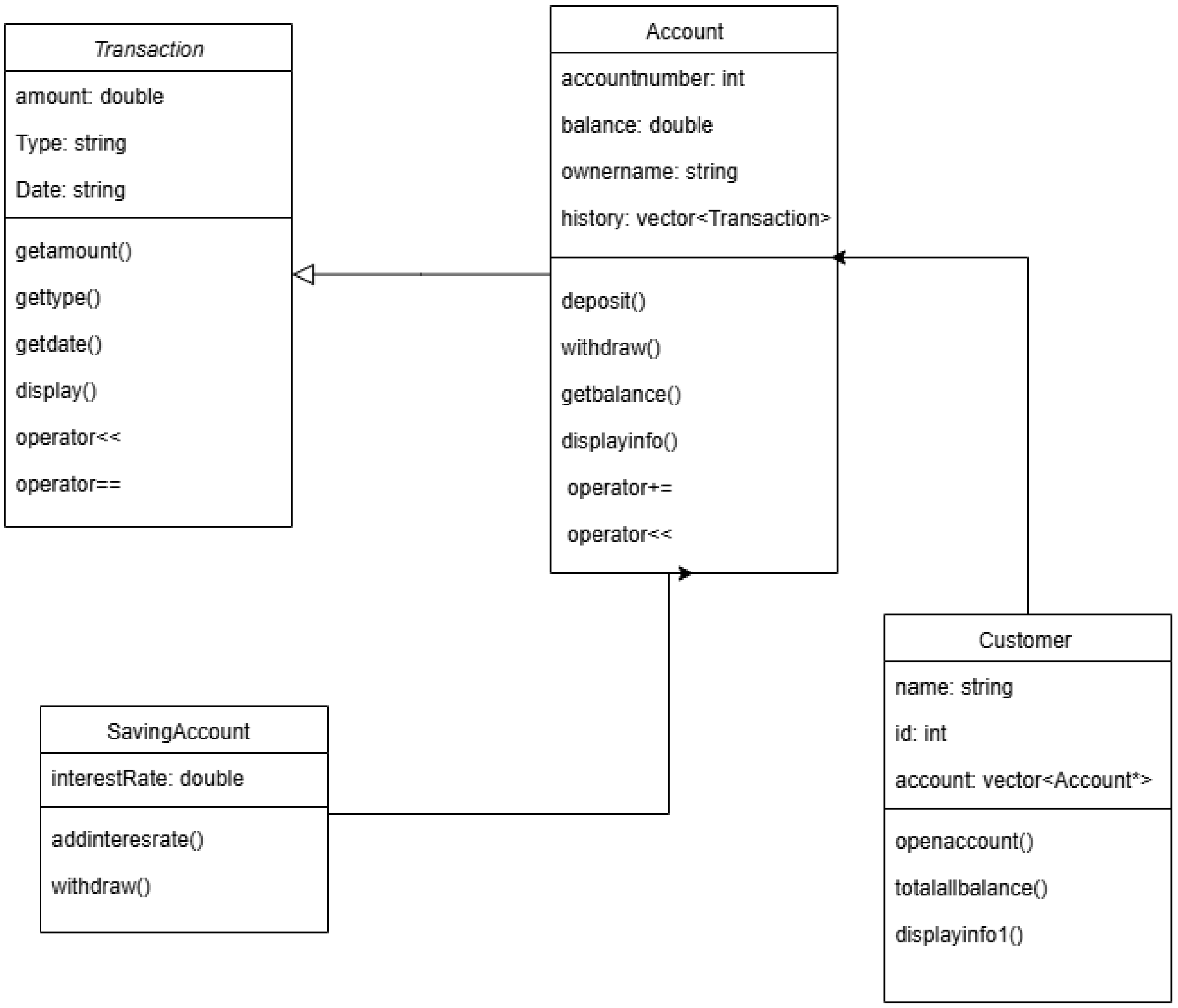
```
--- Test deposit ---

Transaction done: Amount: 200 Type: deposit Date: today

accountNumber: 1001

Balance: 700

ownerName: Alice


--- Test withdraw (success) ---

Transaction done: Amount: 100 Type: withdraw Date: today

accountNumber: 1001
```

```
Balance: 600

ownerName: Alice


--- Test withdraw (fail) ---

Not enough balance to withdraw.

Withdraw failed: not enough balance.

accountNumber: 1001

Balance: 600

ownerName: Alice


--- Test operator += ---

Transaction done: Amount: 300 Type: deposit Date: today

Transaction done: Amount: 200 Type: withdraw Date: today

accountNumber: 1001

Balance: 700

ownerName: Alice


--- Test SavingAccount interest ---

Transaction done: Amount: 50 Type: interest Date: today

accountNumber: 1002

Balance: 1050

ownerName: Alice


--- Test SavingAccount withdraw ---
```

```
Transaction done: Amount: 200 Type: withdraw Date: today
Transaction done: Amount: 2 Type: fee Date: today
accountNumber: 1002
Balance: 848
ownerName: Alice

--- Test Transaction comparison ---
two accounts are similar

--- Customer info ---
Customer: Alice
 | ID: 101
Accounts:
accountNumber: 1001
Balance: 700
ownerName: Alice

accountNumber: 1002
Balance: 848
ownerName: Alice

Total Balance: 1548
```
👉 Explanation:

---

5. UML Diagrams



Class Diagram

- Classes: Transaction, Account, SavingAccount, Customer.
- Relationships:
  - SavingAccount **inherits** from Account.
  - Account **contains** Transaction (composition).
  - Customer **has** multiple Account objects.

(Represented by rectangles with attributes/methods, arrows for inheritance and composition).

Sequence Diagram (for deposit operation)

1. main calls acc1->deposit(200).

2. deposit updates the balance.
3. Creates a Transaction object.
4. Adds it to history.
5. Prints "Transaction done...".

---

6. Use of LLM Tools

- Generating initial ideas for object-oriented design and UML diagrams.
- Providing examples of C++ code for class structure, operator overloading, and inheritance.
- Suggesting documentation structure (OOA analysis, class design, test cases).