CSCI 1120 Introduction to Computing Using C++

Tutorial 9: Account Inheritance Hierarchy

YU, Xiaotian
SHB 1024
xtyu@cse.cuhk.edu.hk

Outline

- Case Study: Scenario
- Problem Analysis
- Inheritance Hierarchy
- Superclass and Subclass
- Test and Client Program
- Summary

Case Study

- A bank try to efficiently represent customers' bank account by using OOP.
- Customers' account usually has two operations
  - Deposit money (called credit)
  - Withdraw money (called debit)
- We discuss two specific types of accounts
  - Savings accounts
  - Checking accounts
- Should we define a class for each account?
  - Customer account, savings account, and checking account

Outline

- Case Study: Scenario
- Problem Analysis
- Inheritance Hierarchy
- Superclass and Subclass
- Test and Client Program
- Summary

Problem analysis (a)

- We find that the member functions and variables among the accounts are overlapped.
- When designing two or more classes that are different but share some common features, we use inheritance.
- The customer account can be viewed as the base class (or, superclass), then the savings account and checking account can be viewed as the derived classes (or, subclass).

Problem analysis (b)

- Base class: *Account*
  - Data member: (type double) to represent the account balance
  - Constructor: to initialize the account balance; and also check the initial balance >= 0, if not satisfied, display an error message
  - Three member functions:
    - credit: add an amount to the current balance
    - debit: withdraw money and ensure that the debit amount should not exceed the current balance. If it does, the balance should be unchanged and display some error message.
    - getBalance: return the current balance

Problem analysis (c)

- Derived class: *SavingsAccount*
  - Inherit the functionality of Account
    - credit
    - debit
    - getBalancce
  - New data member: (type double) interest rate
    - Because save money in bank
  - New public member function: calculateInterest()
    - return a double indicating the amount of interest earned by an account.

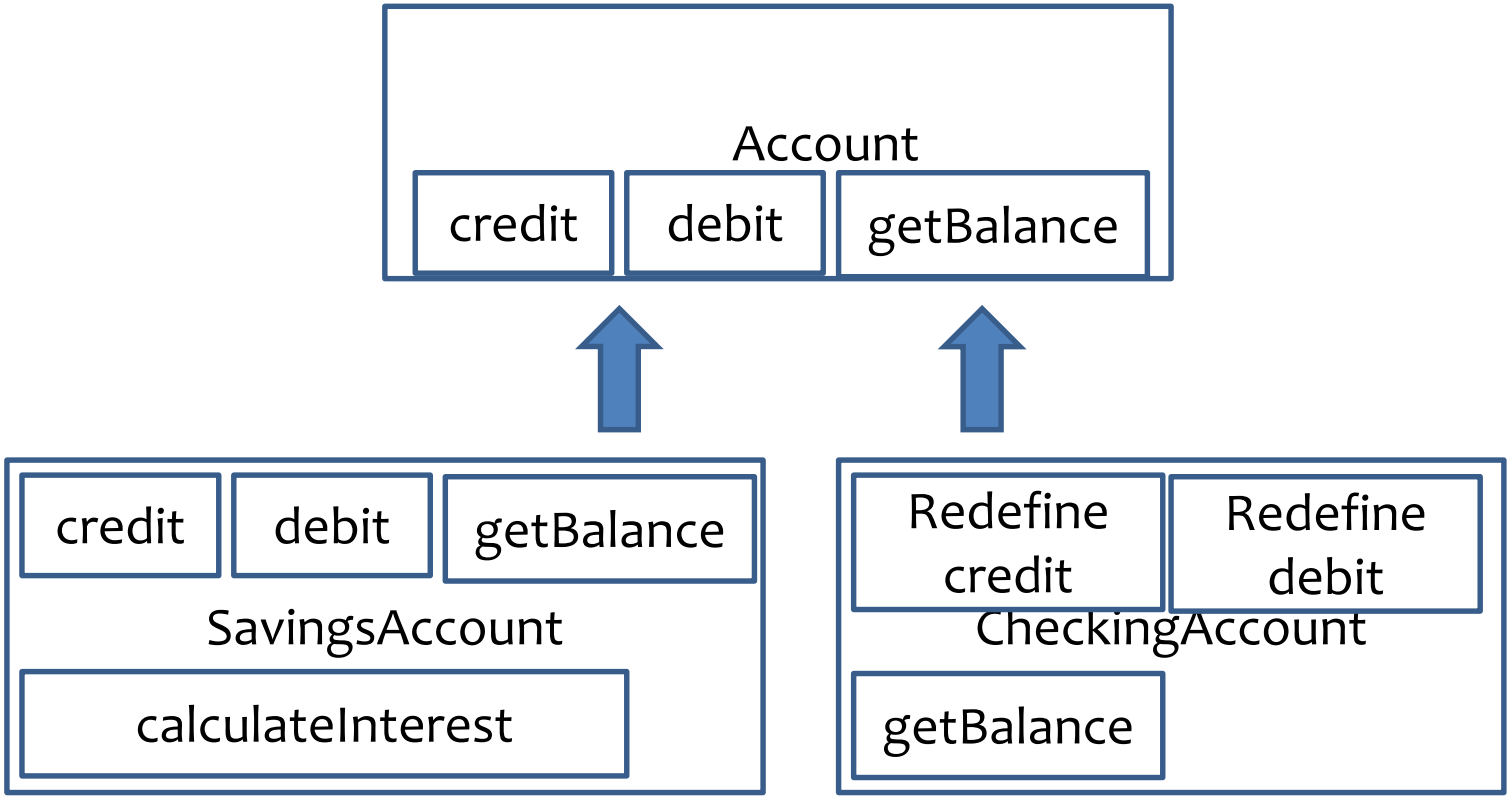
Problem analysis (d)

- Derived class: *CheckingAccount*
  - Inherit the functionality of Account
    - getBalancce
  - Further consider transactions fee in accounts
    - New data member: (type double) represent the fee charged per transaction
  - Redefine credit and debit to subtract fee
    - credit: cost fee each time
    - debit: cost fee when withdraw succeeds

Outline

- Case Study: Scenario
- Problem Analysis
- Inheritance Hierarchy
- Superclass and Subclass
- Test and Client Program
- Summary

Inheritance Hierarchy



Outline

- Case Study: Scenario
- Problem Analysis
- Inheritance Hierarchy
- Superclass and Subclass
- Test and Client Program
- Summary

Superclass: header file

- The header file can be shown as follows.
- Two operations
  - Save money
  - Withdraw money
- Get account balance

```
class Account {
public:
    Account(double money);  // constructor
    void credit(double money); // to save some money in accountBalance
    bool debit(double money);  // to withdraw some money in accountBalance
    double getBalance();       // obtain the accountBalance via method
private:
    double accountBalance;
};
```

Superclass: class file (a)

- Constructor: check the initialization value

```
Account::Account(double money) {
    Account::accountBalance = money;
    if (Account::accountBalance < 0) {  // justify the initial balance
        Account::accountBalance = 0;
        cout << "The initial balance was invalid." << endl; // output error message
    }
}
```

Account::accountBalance and accountBalance are both acceptable

- Save money

```
void Account::credit(double money) {
    Account::accountBalance = Account::accountBalance + money;  // add amount of money
}
```

Superclass: class file (b)

- Withdraw money: check if there is enough money

```
bool Account::debit(double money) {
    if (Account::accountBalance < money) {                // justify the withdraw amount
        cout << "Debit amount exceeded account balancce." << endl; // output error message
        return false;
    }
    else {
        Account::accountBalance = Account::accountBalance - money;  // subtract the amount
        return true;
    }
}
```

Why bool function?

→ to check whether withdraw succeeds or not

- Get account balance

```
double Account::getBalance() {
    return Account::accountBalance;
}
```

Subclass-saving: header file

- The header file should add superclass header file.
- The member functions in superclass are all inherited by default.
- Add some new method and data member

```
#include <iostream>
#include "Account.h"
using namespace std;

class SavingsAccount:public Account{
public:
    SavingsAccount(double money, double rate);  // constructor for SavingsAccount
    double calculateInterest();                 // new member function
private:
    double interestRate;                        // new private data member
};
```

Subclass-saving: class file

- The class file should add subclass header file.
- New constructor

```
SavingsAccount::SavingsAccount(double money, double rate):Account(money){
    SavingsAccount::interestRate = rate;                    // constructor and initialize the interest rate
}
```

from superclass

- Calculate the interest

```
double SavingsAccount::calculateInterest() {
    return getBalance() * SavingsAccount::interestRate / 100;  // calculate the interest
}
```

Can we use accountBalance or Account::accountBalance here?

→ No. Because subclass cannot access the private members in superclass.

Subclass-checking: header file

- The header file should add superclass header file.
- You can also redefine some member functions based on the superclass.

```
class CheckingAccount:public Account{
public:
    CheckingAccount(double money, double fee);  // constructor for CheckingAccount
    void credit(double money);          // redefine the credit member function in the superclass
    bool debit(double money);           // redefine the debit member function in the superclass

private:
    double feePerTrans;                         // new data member
};
```

Subclass-checking: class file

- The class file should add subclass header file.
- New constructor

```
CheckingAccount::CheckingAccount(double money, double fee):Account(money) {
    CheckingAccount::feePerTrans = fee;                    // constructor and initialize the fee
}
```

- How to update the account balance with fee?

```
void CheckingAccount::credit(double money) {                    // redefine the credit function
    Account::credit(money - CheckingAccount::feePerTrans);
}
```

By using the Account::credit() in superclass.

```
bool CheckingAccount::debit(double money) {                    // redefine the debit function
    if (Account::debit(money)) {
        Account::debit(CheckingAccount::feePerTrans);
        return true;
    }
    else {
        return false;
    }
}
```

By using the Account::debit() in superclass.

Outline

- Case Study: Scenario
- Problem Analysis
- Inheritance Hierarchy
- Superclass and Subclass
- Test and Client Program
- Summary

Test - 1

- Base class - a

```
int main() {

    Account consumerAccount(2);
    cout << "accountBalance " << consumerAccount.getBalance() << endl;
    return 0;
```

```
accountBalance 2
```

```
int main() {

    Account consumerAccount(-2);
    cout << "accountBalance " << consumerAccount.getBalance() << endl;
    return 0;
```

```
C:\wind
The initial balance was invalid.
accountBalance 0
```

Test - 2

- Base class - b

```
int main() {

    Account consumerAccount(5);
    cout << "accountBalance " << consumerAccount.getBalance() << endl;
    consumerAccount.credit(4);
    cout << "accountBalance " << consumerAccount.getBalance() << endl;
    consumerAccount.debit(5);
    cout << "accountBalance " << consumerAccount.getBalance() << endl;
    return 0;
```

```
accountBalance 5
accountBalance 9
accountBalance 4
```

Test - 3

- SavingsAccount

```
int main() {

    SavingsAccount savingsAccount(1.2, 10);
    cout << "interest: " << savingsAccount.calculateInterest() << endl;
    savingsAccount.credit(5);
    cout << "accountBalance " << savingsAccount.getBalance() << endl;
    savingsAccount.debit(1);
    cout << "accountBalance " << savingsAccount.getBalance() << endl;
    cout << "interest: " << savingsAccount.calculateInterest() << endl;

    return 0;
```

```
interest: 0.12
accountBalance 6.2
accountBalance 5.2
interest: 0.52
```

Test - 4

- CheckingAccount

```
int main() {

    CheckingAccount checkingAcount(1.2, 0.5);
    cout << "accountBalance " << checkingAcount.getBalance() << endl;
    checkingAcount.credit(0.8);
    cout << "accountBalance " << checkingAcount.getBalance() << endl;
    checkingAcount.debit(5);
    cout << "accountBalance " << checkingAcount.getBalance() << endl;
    checkingAcount.credit(3);
    cout << "accountBalance " << checkingAcount.getBalance() << endl;
    checkingAcount.debit(2);
    cout << "accountBalance " << checkingAcount.getBalance() << endl;

    return 0;
}
```

C:\windows\syste

```
accountBalance 1.2
accountBalance 1.5
Debit amount exceeded account balancce.
accountBalance 1.5
accountBalance 4
accountBalance 1.5
```

Outline

- Case Study: Scenario
- Problem Analysis
- Inheritance Hierarchy
- Superclass and Subclass
- Test and Client Program
- Summary

Summary

- For inheritance, firstly you should be clear what is the base class.
- The relationship between superclass and subclass should be analyzed, e.g., via diagram.
- For the subclass, you should know what is new and what can be inherited via superclass.
- You are recommended to debug/test the superclass and then subclass.
- Use the same function name in subclass as in the superclass.
- Inheritance makes programming efficiently.

Thank You!