**Lab Exercises**

Class: 软件1803 Name: 苏博文 StuID:\_201816040327\_\_\_\_

Date: 2018.12.08 Score:

**Lab Exercise 1 — Account Hierarchy**

**I Lab Objectives**

In this lab, you will practice:

1. Using inheritance to create an account hierarchy that includes an Account class, a SavingsAccount class and a CheckingAccount class.
2. Using private data members to limit access to data members.
3. Redefining base-class member functions in a derived class.

**II Description of the Problem(译文见教材P387 11.10)**

Create an inheritance hierarchy that a bank might use to represent customers’ bank accounts. All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts. More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold. Checking accounts, on the other hand, charge a fee per transaction (i.e., credit or debit).

Create an inheritance hierarchy containing base class Account and derived classes SavingsAccount and CheckingAccount that inherit from class Account. Base class Account should include one data member of type double to represent the account balance. The class should provide a constructor that receives an initial balance and uses it to initialize the data member. The constructor should validate the initial balance to ensure that it is greater than or equal to 0.0. If not, the balance should be set to 0.0 and the constructor should display an error message, indicating that the initial balance was invalid. The class should provide three member functions. Member function credit should add an amount to the current balance. Member function debit should withdraw money from the Account and ensure that the debit amount does not exceed the Account’s balance. If it does, the balance should be left unchanged and the function should print the message "Debit amount exceeded account balance." Member function getBalance should return the current balance.

Derived class SavingsAccount should inherit the functionality of an Account, but also include a data member of type double indicating the interest rate (percentage) assigned to the Account. SavingsAccount’s constructor should receive the initial balance, as well as an initial value for the SavingsAccount’s interest rate. SavingsAccount should provide a public member function calculateInterest that returns a double indicating the amount of interest earned by an account. Member function calculateInterest should determine this amount by multiplying the interest rate by the account balance. [Note: SavingsAccount should inherit member functions credit and debit as is without redefining them.]

Derived class CheckingAccount should inherit from base class Account and include an additional data member of type double that represents the fee charged per transaction. CheckingAccount’s constructor should receive the initial balance, as well as a parameter indicating a fee amount. Class CheckingAccount should redefine member functions credit and debit so that they subtract the fee from the account balance whenever either transaction is performed successfully. CheckingAccount’s versions of these functions should invoke the base-class Account version to perform the updates to an account balance. CheckingAccount’s debit function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance). [*Hint*: Define Account’s debit function so that it returns a bool indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.]

After defining the classes in this hierarchy, write a program that creates objects of each class and tests their member functions. Add interest to the SavingsAccount object by first invoking its calculateInterest function, then passing the returned interest amount to the object’s credit function.

**III Sample Output**



**IV Problem-Solving Tips**

1. Each derived class constructor, SavingsAccount and CheckingAccount, should call the Account constructor explicitly.
2. Do not use the debit member function inside the chargeFee member function, because the debit member function would then call the chargeFee member function, leading to infinite recursion. Instead use the inherited *get* and *set* functions for the account balance.

**V Your Solution**

**double** interestEarned;

interestEarned = account2.calculateInterest();

account2.credit(interestEarned);

cout << "\nNew account2 balance: $" << account2.getBalance() << endl;

CheckingAccount::CheckingAccount(**double** k,**double** n)

:Account(k)

{

transactionfee=n;

}

**void** CheckingAccount::credit(**double** k)

{

Account::credit(k);

}

**void** CheckingAccount::debit(**double** n)

{

Account::debit(n);

}

SavingAccount::SavingAccount(**double** k,**double** n)

:Account(k)

{

interestrate=k;

}

// return the amount of interest earned

/\* Write the calculateInterest member function to return the

interest based on the current balance and interest rate \*/

**double** SavingAccount::calculateInterest()

{

**double** itst;

itst=Account::getBalance()\*interestrate;

**return** itst;

}

**class** SavingAccount:**public** Account

{

**public**:

// constructor initializes balance and interest rate

/\* Declare a two-parameter constructor for SavingsAccount \*/

SavingAccount(**double**=0.0,**double**=0.0);

/\* Declare member function calculateInterest \*/

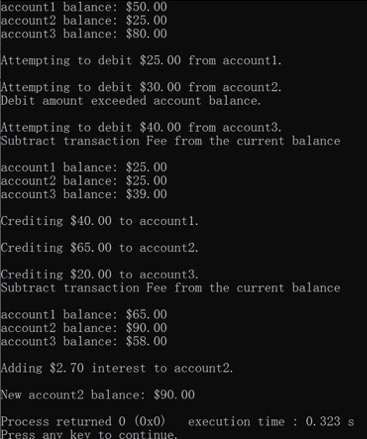
**double** calculateInterest();

**private**:

/\* Declare data member interestRate \*/

**double** interestrate;

}; // end class SavingsAccoun；



**Lab Exercise 2 — Composition**

**I Lab Objectives**

In this lab, you will practice:

1. Using composition to incorporate one class’s members into another class.

The follow-up question and activity also will give you practice:

1. Comparing inheritance and composition.

**II Description of the Problem (译文见P386 11.3)**

Many programs written with inheritance could be written with composition instead, and vice versa. Rewrite class BasePlusCommissionEmployee of the CommissionEmploy ee–BasePlusCommissionEmployee hierarchy to use composition rather than inheritance.

**III Sample Output**



**IV Problem-Solving Tips**

1. To implement BasePlusCommissionEmployee using composition, include a ComissionEmployee object as a data member in the BasePlusCommission Employee class.
2. To access a member of CommissionEmployee inside a member function of BasePlusCommissionEmployee, it must be preceded by the name of the CommissionEmployee object and the dot operator.
3. Most of BasePlusCommissionEmployee’s member functions will be implemented by simply calling the same member function from the CommissionEmployee object; this is known as “delegation.”

**V Your Solution**

:n(first, last, ssn, sales, rate)

{

}

**void** BasePlusCommissionEmployee::setFirstName( **const** string &first )

{

/\* Call commissionEmployee's setFirstName function \*/

//µ˜”√commonemployeeµƒsetFirstName∫Ø ˝

n.setFirstName(first);

} // end function setFirstName

// return commission employee's first name

string BasePlusCommissionEmployee::getFirstName() **const**

{

/\* Call commissionEmployee's getFirstName function \*/

//µ˜”√commissionEmployeeµƒgetFirstName∫Ø ˝

**return** n.getFirstName();

} // end function getFirstName

// set commission employee's last name

**void** BasePlusCommissionEmployee::setLastName( **const** string &last )

{

/\* Call commissionEmployee's setLastName function \*/

n.setLastName(last);

} // end function setLastName

// return commission employee's last name

string BasePlusCommissionEmployee::getLastName() **const**

{

/\* Call commissionEmployee's getLastName function \*/

**return** n.getLastName();

} // end function getLastName

// set commission employee's social security number

**void** BasePlusCommissionEmployee::setSocialSecurityNumber(**const** string &ssn )

{

/\* Call commissionEmployee's setSocialSecurity function \*/

n.setSocialSecurityNumber(ssn);

} // end function setSocialSecurityNumber

// return commission employee's social security number

string BasePlusCommissionEmployee::getSocialSecurityNumber() **const**

{

/\* Call commissionEmployee's getSocialSecurity function \*/

**return** n.getSocialSecurityNumber();

} // end function getSocialSecurityNumber

// set commission employee's gross sales amount

**void** BasePlusCommissionEmployee::setGrossSales( **double** sales )

{

/\* Call commissionEmployee's setGrossSales function \*/

n.setGrossSales(sales);

} // end function setGrossSales

// return commission employee's gross sales amount

**double** BasePlusCommissionEmployee::getGrossSales() **const**

{

/\* Call commissionEmployee's getGrossSales function \*/

**return** n.getGrossSales();

} // end function getGrossSales

// set commission employee's commission rate

**void** BasePlusCommissionEmployee::setCommissionRate( **double** rate )

{

/\* Call commissionEmployee's setCommissionRate function \*/

n.setCommissionRate(rate);

} // end function setCommissionRate

// return commission employee's commission rate

**double** BasePlusCommissionEmployee::getCommissionRate() **const**

{

/\* Call commissionEmployee's getCommissionRate function \*/

**return** n.getCommissionRate();

} // end function getCommissionRate

// set base salary

**void** BasePlusCommissionEmployee::setBaseSalary( **double** salary )

{

baseSalary = ( salary < 0.0 ) ? 0.0 : salary;

} // end function setBaseSalary

// return base salary

**double** BasePlusCommissionEmployee::getBaseSalary() **const**

{

**return** baseSalary;

} // end function getBaseSalary

// calculate earnings

**double** BasePlusCommissionEmployee::earnings() **const**

{

//return getBaseSalary() + ;

/\* Call commissionEmployee's earnings function \*/

//µ˜”√commissionEmployeµƒearningπ¶ƒ‹∫Ø ˝

**return** n.earnings();

} // end function earnings

// print BasePlusCommissionEmployee object

**void** BasePlusCommissionEmployee::print() **const**

{

cout << "base-salaried ";

// invoke composed CommissionEmployee object's print function

/\* Call commissionEmployee's print function \*/

**return** n.print();

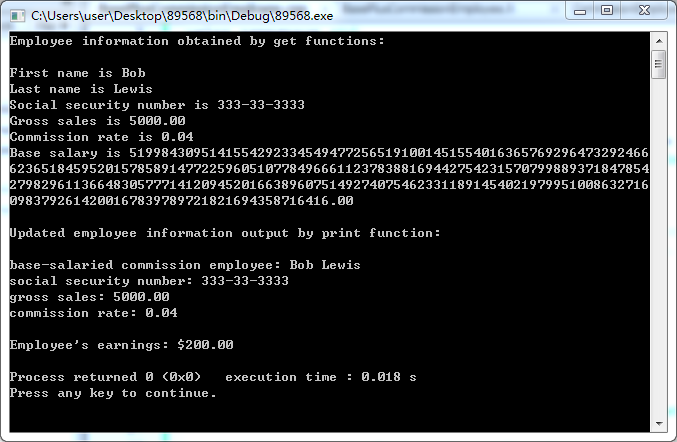
cout << "\nbase salary: " << getBaseSalary();

} // end function print

/\* Write a declaration for a CommissionEmployee

data member \*/

CommissionEmployee n;



**VI Follow-Up Questions and Activities**

1. Assess the relative merits of the two approaches for designing classes Commission-Employee and BasePlusCommissionEmployee, as well as for object-oriented programs in general. Which approach is more natural? Why?

　　①：当前对象只能通过所包含的那个对象去调用其方法，所以所包含的对象的内部细节对当前对象时不可见的。

　　②：当前对象与包含的对象是一个低耦合关系，如果修改包含对象的类中代码不需要修改当前对象类的代码。

　　③：当前对象可以在运行时动态的绑定所包含的对象。可以通过set方法给所包含对象赋值。