**Lab Exercises**

Class: 软件1801 Name: 渠超众 StuID: 201816040109

Date: 2019-11-27 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**

|  |
| --- |
| // Lab 1: Account.h  // Definition of Account class.  #ifndef ACCOUNT\_H  #define ACCOUNT\_H  class Account  {  public:  Account(double); // constructor initializes balance  void credit(double); // add an amount to the account balance  bool debit(double); // subtract an amount from the account balance  void setBalance(double); // sets the account balance  double getBalance(); // return the account balance  private:  double balance; // data member that stores the balance  }; // end class Account  #endif |

|  |
| --- |
| // Lab 1: Account.cpp  // Member-function definitions for class Account.  #include <iostream>  using namespace std;  #include "Account.h" // include definition of class Account  // Account constructor initializes data member balance  Account::Account(double initialBalance)  {  // if initialBalance is greater than or equal to 0.0, set this value  // as the balance of the Account  if (initialBalance >= 0.0)  balance = initialBalance;  else // otherwise, output message and set balance to 0.0  {  cout << "Error: Initial balance cannot be negative." << endl;  balance = 0.0;  } // end if...else  } // end Account constructor  // credit (add) an amount to the account balance  void Account::credit(double amount)  {  balance = balance + amount; // add amount to balance  } // end function credit  // debit (subtract) an amount from the account balance  // return bool indicating whether money was debited  bool Account::debit(double amount)  {  if (amount > balance) // debit amount exceeds balance  {  cout << "Debit amount exceeded account balance." << endl;  return false;  } // end if  else // debit amount does not exceed balance  {  balance = balance - amount;  return true;  } // end else  } // end function debit  // set the account balance  void Account::setBalance(double newBalance)  {  balance = newBalance;  } // end function setBalance  // return the account balance  double Account::getBalance()  {  return balance;  } // end function getBalance |

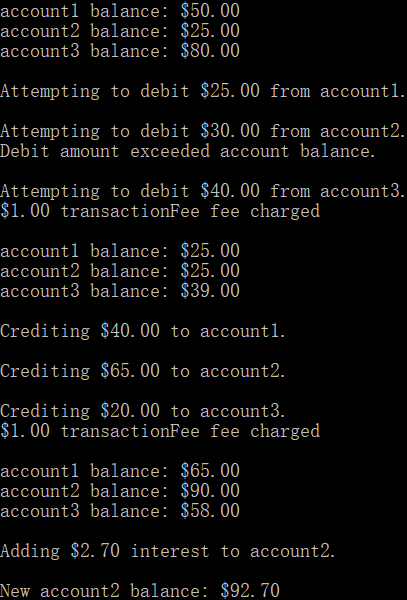
|  |
| --- |
| // Lab 1: CheckingAccount.h  // Definition of CheckingAccount class.  #ifndef CHECKING\_H  #define CHECKING\_H  #include "Account.h"  /\* Write a directive to include the Account header file \*/  class CheckingAccount : public Account  /\* Write a line to have class CheckingAccount inherit publicly from Account \*/  {  public:  // constructor initializes balance and transaction fee  /\* Declare a two-argument constructor for CheckingAccount \*/  CheckingAccount(double, double);  /\* Redeclare member function credit, which will be redefined \*/  void credit(double);  bool debit(double);  /\* Redeclare member function debit, which will be redefined \*/  private:  /\* Declare data member transactionFee \*/  double transactionFee;  void chargeFee();  // utility function to charge fee  /\* Declare member function chargeFee \*/  }; // end class CheckingAccount  #endif |

|  |
| --- |
| // Lab 1: CheckingAccount.cpp  // Member-function definitions for class CheckingAccount.  #include <iostream>  using namespace std;  #include "CheckingAccount.h" // CheckingAccount class definition  // constructor initializes balance and transaction fee  /\* Write the CheckingAccount constructor to call the Account constructor  and validate and set the transaction fee value \*/  CheckingAccount::CheckingAccount(double a, double b) : Account(a)  {  transactionFee = b;  }  // credit (add) an amount to the account balance and charge fee  /\* Write the credit member function to call Account's credit function  and then charge a fee \*/  void CheckingAccount::credit(double amount)  {  Account::credit(amount);  if (getBalance() >= transactionFee)  {  chargeFee();  }  else  {  cout << "Transaction fee exceeded account balance while crediting" << endl;  Account::debit(amount);  }  }  // debit (subtract) an amount from the account balance and charge fee  /\* Write the debit member function to call Account's debit function  and then charge a fee if it returned true\*/  bool CheckingAccount::debit(double amount)  {  bool flag = Account::debit(amount);  if (flag == true)  {  if (getBalance() < transactionFee)  {  cout << "Transaction fee exceeded account balance while debiting\n";  return false;  }  else  {  chargeFee();  return true;  }  }  else  {  return false;  }  }  // subtract transaction fee  /\* Write the chargeFee member function to subtract transactionFee  from the current balance and display a message \*/  void CheckingAccount::chargeFee()  {  Account::debit(transactionFee);  cout << "$1.00 transactionFee fee charged" << endl;  } |

|  |
| --- |
| // Lab 1: SavingsAccount.h  // Definition of SavingsAccount class.  #ifndef SAVINGS\_H  #define SAVINGS\_H  #include "Account.h"  /\* Write a directive to include the Account header file \*/  class SavingsAccount : public Account  /\* Write a line to have class SavingsAccount inherit publicly from Account \*/  {  public:  // constructor initializes balance and interest rate  /\* Declare a two-parameter constructor for SavingsAccount \*/  SavingsAccount(double, double);  double calculateInterest();  /\* Declare member function calculateInterest \*/  private:  /\* Declare data member interestRate \*/  double interestRate;  }; // end class SavingsAccount  #endif |

|  |
| --- |
| // Lab 1: SavingsAccount.cpp  // Member-function definitions for class SavingsAccount.  #include "SavingsAccount.h" // SavingsAccount class definition  // constructor initializes balance and interest rate  /\* Write the SavingsAccount constructor to call the Account constructor  and validate and set the interest rate value \*/  SavingsAccount::SavingsAccount(double balance, double rate) : Account(balance)  {  interestRate = rate;  }  // return the amount of interest earned  /\* Write the calculateInterest member function to return the  interest based on the current balance and interest rate \*/  double SavingsAccount::calculateInterest()  {  return interestRate \* getBalance();  } |

|  |
| --- |
| // Lab 1: bankAccounts.cpp  // Test program for Account hierarchy.  #include <iostream>  #include <iomanip>  using namespace std;  #include "Account.h" // Account class definition  #include "SavingsAccount.h" // SavingsAccount class definition  #include "CheckingAccount.h" // CheckingAccount class definition  int main()  {  Account account1(50.0); // create Account object  SavingsAccount account2(25.0, 0.03); // create SavingsAccount object  CheckingAccount account3(80.0, 1.0); // create CheckingAccount object  cout << fixed << setprecision(2);  // display initial balance of each object  cout << "account1 balance: $" << account1.getBalance() << endl;  cout << "account2 balance: $" << account2.getBalance() << endl;  cout << "account3 balance: $" << account3.getBalance() << endl;  cout << "\nAttempting to debit $25.00 from account1." << endl;  account1.debit(25.0); // try to debit $25.00 from account1  cout << "\nAttempting to debit $30.00 from account2." << endl;  account2.debit(30.0); // try to debit $30.00 from account2  cout << "\nAttempting to debit $40.00 from account3." << endl;  account3.debit(40.0); // try to debit $40.00 from account3  // display balances  cout << "\naccount1 balance: $" << account1.getBalance() << endl;  cout << "account2 balance: $" << account2.getBalance() << endl;  cout << "account3 balance: $" << account3.getBalance() << endl;  cout << "\nCrediting $40.00 to account1." << endl;  account1.credit(40.0); // credit $40.00 to account1  cout << "\nCrediting $65.00 to account2." << endl;  account2.credit(65.0); // credit $65.00 to account2  cout << "\nCrediting $20.00 to account3." << endl;  account3.credit(20.0); // credit $20.00 to account3  // display balances  cout << "\naccount1 balance: $" << account1.getBalance() << endl;  cout << "account2 balance: $" << account2.getBalance() << endl;  cout << "account3 balance: $" << account3.getBalance() << endl;  // add interest to SavingsAccount object account2  /\* Declare a variable interestEarned and assign it the interest  account2 should earn \*/  cout << "\nAdding $" << account2.calculateInterest() << " interest to account2."  << endl;  /\* Write a statement to credit the interest to account2's balance \*/  account2.credit(account2.calculateInterest());  cout << "\nNew account2 balance: $" << account2.getBalance() << endl;  } // end main |



**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**

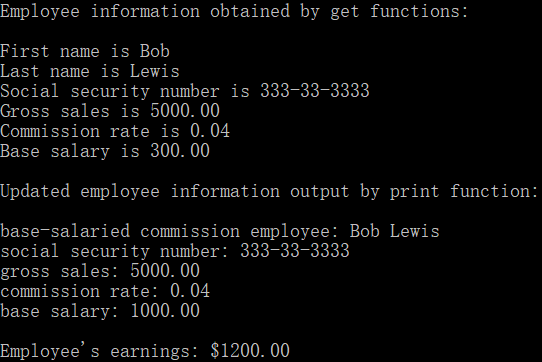
|  |
| --- |
| // Lab 2: BasePlusCommissionEmployee.h  // BasePlusCommissionEmployee class using composition.  #ifndef BASEPLUS\_H  #define BASEPLUS\_H  #include <string> // C++ standard string class  using namespace std;  #include "CommissionEmployee.h" // CommissionEmployee class definition  class BasePlusCommissionEmployee  {  public:  BasePlusCommissionEmployee(const string &, const string &,  const string &, double = 0.0, double = 0.0, double = 0.0);  void setFirstName(const string &); // set first name  string getFirstName() const; // return first name  void setLastName(const string &); // set last name  string getLastName() const; // return last name  void setSocialSecurityNumber(const string &); // set SSN  string getSocialSecurityNumber() const; // return SSN  void setGrossSales(double); // set gross sales amount  double getGrossSales() const; // return gross sales amount  void setCommissionRate(double); // set commission rate  double getCommissionRate() const; // return commission rate  void setBaseSalary(double); // set base salary  double getBaseSalary() const; // return base salary  double earnings() const; // calculate earnings  void print() const; // print BasePlusCommissionEmployee object  private:  double baseSalary; // base salary  /\* Write a declaration for a CommissionEmployee  data member \*/  CommissionEmployee ce;  }; // end class BasePlusCommissionEmployee  #endif |

|  |
| --- |
| // Lab 2: BasePlusCommissionEmployee.cpp  // Member-function definitions of class BasePlusCommissionEmployee  // using composition.  #include <iostream>  using namespace std;  // BasePlusCommissionEmployee class definition  #include "BasePlusCommissionEmployee.h"  // constructor  BasePlusCommissionEmployee::BasePlusCommissionEmployee(  const string &first, const string &last, const string &ssn,  double sales, double rate, double salary)  // initialize composed object  : ce(first, last, ssn, sales, rate) /\* Initialize the commissionEmployee data member,  pass (first, last, ssn, sales, rate) to its constructor \*/  {  setBaseSalary(salary); // validate and store base salary  } // end BasePlusCommissionEmployee constructor  // set commission employee's first name  void BasePlusCommissionEmployee::setFirstName(const string &first)  {  ce.setFirstName(first);  /\* Call commissionEmployee's setFirstName function \*/  } // end function setFirstName  // return commission employee's first name  string BasePlusCommissionEmployee::getFirstName() const  {  return ce.getFirstName();  /\* Call commissionEmployee's getFirstName function \*/  } // end function getFirstName  // set commission employee's last name  void BasePlusCommissionEmployee::setLastName(const string &last)  {  ce.setLastName(last);  /\* Call commissionEmployee's setLastName function \*/  } // end function setLastName  // return commission employee's last name  string BasePlusCommissionEmployee::getLastName() const  {  return ce.getLastName();  /\* Call commissionEmployee's getLastName function \*/  } // end function getLastName  // set commission employee's social security number  void BasePlusCommissionEmployee::setSocialSecurityNumber(  const string &ssn)  {  ce.setSocialSecurityNumber(ssn);  /\* Call commissionEmployee's setSocialSecurity function \*/  } // end function setSocialSecurityNumber  // return commission employee's social security number  string BasePlusCommissionEmployee::getSocialSecurityNumber() const  {  return ce.getSocialSecurityNumber();  /\* Call commissionEmployee's getSocialSecurity function \*/  } // end function getSocialSecurityNumber  // set commission employee's gross sales amount  void BasePlusCommissionEmployee::setGrossSales(double sales)  {  ce.setGrossSales(sales);  /\* Call commissionEmployee's setGrossSales function \*/  } // end function setGrossSales  // return commission employee's gross sales amount  double BasePlusCommissionEmployee::getGrossSales() const  {  return ce.getGrossSales();  /\* Call commissionEmployee's getGrossSales function \*/  } // end function getGrossSales  // set commission employee's commission rate  void BasePlusCommissionEmployee::setCommissionRate(double rate)  {  ce.setCommissionRate(rate);  /\* Call commissionEmployee's setCommissionRate function \*/  } // end function setCommissionRate  // return commission employee's commission rate  double BasePlusCommissionEmployee::getCommissionRate() const  {  return ce.getCommissionRate();  /\* Call commissionEmployee's getCommissionRate function \*/  } // 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() + ce.earnings();  /\* Call commissionEmployee's earnings function \*/;  } // 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 \*/  ce.print();  cout << "\nbase salary: " << getBaseSalary();  } // end function print |

|  |
| --- |
| // Lab 2: CommissionEmployee.h  // CommissionEmployee class definition represents a commission employee.  #ifndef COMMISSION\_H  #define COMMISSION\_H  #include <string> // C++ standard string class  using namespace std;  class CommissionEmployee  {  public:  CommissionEmployee(const string &, const string &, const string &,  double = 0.0, double = 0.0);  void setFirstName(const string &); // set first name  string getFirstName() const; // return first name  void setLastName(const string &); // set last name  string getLastName() const; // return last name  void setSocialSecurityNumber(const string &); // set SSN  string getSocialSecurityNumber() const; // return SSN  void setGrossSales(double); // set gross sales amount  double getGrossSales() const; // return gross sales amount  void setCommissionRate(double); // set commission rate (percentage)  double getCommissionRate() const; // return commission rate  double earnings() const; // calculate earnings  void print() const; // print CommissionEmployee object  private:  string firstName;  string lastName;  string socialSecurityNumber;  double grossSales; // gross weekly sales  double commissionRate; // commission percentage  }; // end class CommissionEmployee  #endif |

|  |
| --- |
| // Lab 2: CommissionEmployee.cpp  // Class CommissionEmployee member-function definitions.  #include <iostream>  using namespace std;  #include "CommissionEmployee.h" // CommissionEmployee class definition  // constructor  CommissionEmployee::CommissionEmployee(  const string &first, const string &last, const string &ssn,  double sales, double rate)  {  firstName = first; // should validate  lastName = last; // should validate  socialSecurityNumber = ssn; // should validate  setGrossSales(sales); // validate and store gross sales  setCommissionRate(rate); // validate and store commission rate  } // end CommissionEmployee constructor  // set first name  void CommissionEmployee::setFirstName(const string &first)  {  firstName = first; // should validate  } // end function setFirstName  // return first name  string CommissionEmployee::getFirstName() const  {  return firstName;  } // end function getFirstName  // set last name  void CommissionEmployee::setLastName(const string &last)  {  lastName = last; // should validate  } // end function setLastName  // return last name  string CommissionEmployee::getLastName() const  {  return lastName;  } // end function getLastName  // set social security number  void CommissionEmployee::setSocialSecurityNumber(const string &ssn)  {  socialSecurityNumber = ssn; // should validate  } // end function setSocialSecurityNumber  // return social security number  string CommissionEmployee::getSocialSecurityNumber() const  {  return socialSecurityNumber;  } // end function getSocialSecurityNumber  // set gross sales amount  void CommissionEmployee::setGrossSales(double sales)  {  grossSales = (sales < 0.0) ? 0.0 : sales;  } // end function setGrossSales  // return gross sales amount  double CommissionEmployee::getGrossSales() const  {  return grossSales;  } // end function getGrossSales  // set commission rate  void CommissionEmployee::setCommissionRate(double rate)  {  commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0;  } // end function setCommissionRate  // return commission rate  double CommissionEmployee::getCommissionRate() const  {  return commissionRate;  } // end function getCommissionRate  // calculate earnings  double CommissionEmployee::earnings() const  {  return commissionRate \* grossSales;  } // end function earnings  // print CommissionEmployee object  void CommissionEmployee::print() const  {  cout << "commission employee: " << firstName << ' ' << lastName  << "\nsocial security number: " << socialSecurityNumber  << "\ngross sales: " << grossSales  << "\ncommission rate: " << commissionRate;  } // end function print |

|  |
| --- |
| // Lab 2: composition.cpp  // Testing class BasePlusCommissionEmployee.  #include <iostream>  #include <iomanip>  using namespace std;  // BasePlusCommissionEmployee class definition  #include "BasePlusCommissionEmployee.h"  int main()  {  // instantiate BasePlusCommissionEmployee object  BasePlusCommissionEmployee  employee("Bob", "Lewis", "333-33-3333", 5000, 0.04, 300);  // set floating-point output formatting  cout << fixed << setprecision(2);  // get commission employee data  cout << "Employee information obtained by get functions: \n"  << "\nFirst name is " << employee.getFirstName()  << "\nLast name is " << employee.getLastName()  << "\nSocial security number is "  << employee.getSocialSecurityNumber()  << "\nGross sales is " << employee.getGrossSales()  << "\nCommission rate is " << employee.getCommissionRate()  << "\nBase salary is " << employee.getBaseSalary() << endl;  employee.setBaseSalary(1000); // set base salary  cout << "\nUpdated employee information output by print function: \n"  << endl;  employee.print(); // display the new employee information  // display the employee's earnings  cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;  } // end main |

****

**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?

我觉得组合比继承更自然，组合低耦合、封装型好，更适合面向对象程序的设计。