**Lab Exercises**

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

Date: 2019.12.15 Score:

**Lab Exercise — Polymorphic Banking**

**I Lab Objectives**

In this lab, you will practice:

1. Creating an Account base class that contains virtual functions and derived classes SavingsAccount and CheckingAccount.
2. Defining virtual functions.
3. Calling virtual functions.
4. Downcasting with a pointer with the dynamic\_cast operator.

**II Description of the Problem (译文见教材P419 12.14)**

Develop a polymorphic banking program using the Account hierarchy created in Exercise 11.10. Create a vector of Account pointers to SavingsAccount and CheckingAccount objects. For each Account in the vector, allow the user to specify an amount of money to withdraw from the Account using member function debit and an amount of money to deposit into the Account using member function credit. As you process each Account, determine its type. If an Account is a SavingsAccount, calculate the amount of interest owed to the Account using member function calculateInterest, then add the interest to the account balance using member function credit. After processing an Account, print the updated account balance obtained by invoking base class member function getBalance.

**III Sample Output**

****

**IV Problem-Solving Tips**

1. To achieve polymorphism, declare the functions that should be called polymorphically as virtual. To indicate a virtual function within a class definition, add “virtual” before the function prototype. When the virtual functions are redefined in a derived class, those member function prototypes should also be preceded by the keyword virtual as a good programming practice.
2. To determine if a pointer to an Account object is actually pointing to a SavingsAccount object, downcast it to a SavingsAccount \* using the dynamic\_cast operator. If the pointer returned by this operation is not the null pointer (i.e., 0) then the object is a SavingsAccount object and that pointer can be used to access members unique to class SavingsAccount.
3. Remember that your compiler may require you to enable run-time type information (RTTI) for this particular project before this program will run correctly.

**V Your Solution**

**class** Account

{

**public**:

Account( **double** ); // constructor initializes balance

/\* Write a function prototype for virtual function credit

为虚函数写一个函数原型\*/

**virtual** **void** credit(**double** );

/\* Write a function prototype for virtual function debit \*/

**virtual** **bool** debit(**double** );

**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

// loop through vector, prompting user for debit and credit amounts

**for** ( size\_t i = 0; i < accounts.size(); i++ )

{

cout << "Account " << i + 1 << " balance: $"

<< /\* Call the getBalance function through Account pointer i \*/accounts[i]->getBalance();

**double** withdrawalAmount = 0.0;

cout << "\nEnter an amount to withdraw from Account " << i + 1

<< ": ";

cin >> withdrawalAmount;

/\* Call the debit function through Account pointer i \*/

**double** depositAmount = 0.0;

cout << "Enter an amount to deposit into Account " << i + 1

<< ": ";

cin >> depositAmount;

/\* Call the credit function through Account pointer i \*/

// downcast pointer

SavingsAccount \*savingsAccountPtr = **dynamic\_cast**<SavingsAccount \*>(accounts[i]);

/\* Write a dynamic\_cast operation to to attempt to downcast

Account pointer i to a SavingsAccount pointer \*/

savingsAccountPtr->calculateInterest();

// if Account is a SavingsAccount, calculate and add interest

**if** ( savingsAccountPtr!=0 /\* Write a test to determine if savingsAccountPtr isn't 0 \*/ )

{

**double** interestEarned = savingsAccountPtr->calculateInterest(); /\* Call member function calculateInterest

through savingsAccountPtr \*/;

cout << "Adding $" << interestEarned << " interest to Account "

<< i + 1 << " (a SavingsAccount)" << endl;

/\* Use the credit function to credit interestEarned to

the SavingsAccount pointed to by savingsAccountPtr\*/

savingsAccountPtr->credit(interestEarned);

} // end if

cout << "Updated Account " << i + 1 << " balance: $"

<< /\* Call the getBalance function through Account pointer i \*/

accounts[i]->getBalance() << "\n\n";

} // end for

