**Assignment 4:**

1. **Assignment Objectives**

In this assignment, you will practice:  
• Creating an **Account** base class that contains **virtual** functions and derived classes **SavingsAccount** and  
**CheckingAccount**.  
• Defining **virtual** functions.  
• Calling **virtual** functions.  
• **Downcasting** with a pointer with the **dynamic**\_**cast** operator.

1. **Description of the Problem**   
   Develop a polymorphic banking program using the **Account** hierarchy. 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**.
2. **Sample Output**

|  |
| --- |
| Account 1 balance: $25.00 Enter an amount to withdraw from Account 1: 15.00 Enter an amount to deposit into Account 1: 10.50 Adding $0.61 interest to Account 1 (a SavingsAccount) Updated Account 1 balance: $21.11 Account 2 balance: $80.00 Enter an amount to withdraw from Account 2: 90.00 Debit amount exceeded account balance. Enter an amount to deposit into Account 2: 45.00 $1.00 transaction fee charged. Updated Account 2 balance: $124.00 Account 3 balance: $200.00 Enter an amount to withdraw from Account 3: 75.50 Enter an amount to deposit into Account 3: 300.00 Adding $6.37 interest to Account 3 (a SavingsAccount) Updated Account 3 balance: $430.87 Account 4 balance: $400.00 Enter an amount to withdraw from Account 4: 56.81 $0.50 transaction fee charged. Enter an amount to deposit into Account 4: 37.83 $0.50 transaction fee charged. Updated Account 4 balance: $380.02 |

1. **Template**

4.1 Code | Account.h

*1 // Lab 1: Account.h  
2 // Definition of Account class.  
3 #ifndef ACCOUNT\_H  
4 #define ACCOUNT\_H  
5 6  
class Account  
7 {  
8 public:  
9 Account( double ); // constructor initializes balance  
10 /\* Write a function prototype for virtual function credit \*/  
11 /\* Write a function prototype for virtual function debit \*/  
12 void setBalance( double ); // sets the account balance  
13 double getBalance(); // return the account balance  
14 private:  
15 double balance; // data member that stores the balance  
16 }; // end class Account  
17  
18 #endif*

4.2 Code | Account.cpp

*1 // Lab 1: Account.cpp  
2 // Member-function definitions for class Account.  
3 #include <iostream>  
4 using std::cout;  
5 using std::endl;  
6 7  
#include "Account.h" // include definition of class Account  
8 9  
// Account constructor initializes data member balance  
10 Account::Account( double initialBalance )  
11 {  
12 // if initialBalance is greater than or equal to 0.0, set this value  
13 // as the balance of the Account  
14 if ( initialBalance >= 0.0 )  
15 balance = initialBalance;  
16 else // otherwise, output message and set balance to 0.0  
17 {  
18 cout << "Error: Initial balance cannot be negative." << endl;  
19 balance = 0.0;  
20 } // end if...else  
21 } // end Account constructor  
22  
23 // credit (add) an amount to the account balance  
24 void Account::credit( double amount )  
25 {  
26 balance = balance + amount; // add amount to balance  
27 } // end function credit  
28  
29 // debit (subtract) an amount from the account balance  
30 // return bool indicating whether money was debited  
31 bool Account::debit( double amount )  
32 {  
33 if ( amount > balance ) // debit amount exceeds balance  
34 {  
35 cout << "Debit amount exceeded account balance." << endl;  
36 return false;  
37 } // end if  
38 else // debit amount does not exceed balance  
39 {  
40 balance = balance - amount;  
41 return true;  
42 } // end else  
43 } // end function debit  
44  
45 // set the account balance  
46 void Account::setBalance( double newBalance )  
47 {  
48 balance = newBalance;  
49 } // end function setBalance  
50  
51 // return the account balance  
52 double Account::getBalance()  
53 {  
54 return balance;  
55 } // end function getBalance*

4.3 Code | SavingsAccount.h

*1 // Lab 1: SavingsAccount.h  
2 // Definition of SavingsAccount class.  
3 #ifndef SAVINGS\_H  
4 #define SAVINGS\_H  
5 6  
#include "Account.h" // Account class definition  
7 8  
class SavingsAccount : public Account  
9 {  
10 public:  
11 // constructor initializes balance and interest rate  
12 SavingsAccount( double, double );  
13  
14 double calculateInterest(); // determine interest owed  
15 private:  
16 double interestRate; // interest rate (percentage) earned by account  
17 }; // end class SavingsAccount  
18  
19 #endif*

4.4 Code | SavingsAccount.cpp

*1 // Lab 1: SavingsAccount.cpp  
2 // Member-function definitions for class SavingsAccount.  
3 #include "SavingsAccount.h" // SavingsAccount class definition  
4 5  
// constructor initializes balance and interest rate  
6 SavingsAccount::SavingsAccount( double initialBalance, double rate )  
7 : Account( initialBalance ) // initialize base class  
8 {  
9 interestRate = ( rate < 0.0 ) ? 0.0 : rate; // set interestRate  
10 } // end SavingsAccount constructor  
11  
12 // return the amount of interest earned  
13 double SavingsAccount::calculateInterest()  
14 {  
15 return getBalance() \* interestRate;  
16 } // end function calculateInterest*

4.5 Code | CheckingAccount.h

*1 // Lab 1: CheckingAccount.h  
2 // Definition of CheckingAccount class.  
3 #ifndef CHECKING\_H  
4 #define CHECKING\_H  
5 6  
#include "Account.h" // Account class definition  
7*

*8  
class CheckingAccount : public Account  
9 {  
10 public:  
11 // constructor initializes balance and transaction fee  
12 CheckingAccount( double, double );  
13*

*14 /\* Write a function prototype for virtual function credit,  
15 which will redefine the inherited credit function \*/  
16 /\* Write a function prototype for virtual function debit,  
17 which will redefine the inherited debit function \*/  
18 private:  
19 double transactionFee; // fee charged per transaction  
20  
21 // utility function to charge fee  
22 void chargeFee();  
23 }; // end class CheckingAccount  
24  
25 #endif*

4.6 Code | CheckingAccount.cpp

*1 // Lab 1: CheckingAccount.cpp  
2 // Member-function definitions for class CheckingAccount.  
3 #include <iostream>  
4 using std::cout;  
5 using std::endl;  
6*

*7  
#include "CheckingAccount.h" // CheckingAccount class definition  
8*

*9  
// constructor initializes balance and transaction fee  
10 CheckingAccount::CheckingAccount( double initialBalance, double fee )  
11 : Account( initialBalance ) // initialize base class  
12 {  
13 transactionFee = ( fee < 0.0 ) ? 0.0 : fee; // set transaction fee  
14 } // end CheckingAccount constructor  
15  
16 // credit (add) an amount to the account balance and charge fee  
17 void CheckingAccount::credit( double amount )  
18 {  
19 Account::credit( amount ); // always succeeds  
20 chargeFee();  
21 } // end function credit  
22  
23 // debit (subtract) an amount from the account balance and charge fee  
24 bool CheckingAccount::debit( double amount )  
25 {  
26 bool success = Account::debit( amount ); // attempt to debit  
27  
28 if ( success ) // if money was debited, charge fee and return true  
29 {  
30 chargeFee();  
31 return true;  
32 } // end if  
33 else // otherwise, do not charge fee and return false  
34 return false;  
35 } // end function debit  
36  
37 // subtract transaction fee  
38 void CheckingAccount::chargeFee()  
39 {  
40 Account::setBalance( getBalance() - transactionFee );*

*41 cout << "$" << transactionFee << " transaction fee charged." << endl;  
42 } // end function chargeFee*

* 1. Code | polymorphicBanking.cpp

*1 // Lab 1: polymorphicBanking.cpp  
2 // Processing Accounts polymorphically.  
3 #include <iostream>  
4 using std::cout;  
5 using std::cin;  
6 using std::endl;  
7*

*8  
#include <iomanip>  
9 using std::setprecision;  
10 using std::fixed;  
11  
12 #include <vector>  
13 using std::vector;  
14  
15 #include "Account.h" // Account class definition  
16 #include "SavingsAccount.h" // SavingsAccount class definition  
17 #include "CheckingAccount.h" // CheckingAccount class definition  
18  
19 int main()  
20 {  
21 // create vector accounts  
22 /\* Write declarations for a vector of four pointers  
23 to Account objects, called accounts \*/  
24  
25 // initialize vector with Accounts  
26 accounts[ 0 ] = new SavingsAccount( 25.0, .03 );  
27 accounts[ 1 ] = new CheckingAccount( 80.0, 1.0 );  
28 accounts[ 2 ] = new SavingsAccount( 200.0, .015 );  
29 accounts[ 3 ] = new CheckingAccount( 400.0, .5 );  
30  
31 cout << fixed << setprecision( 2 );  
32  
33 // loop through vector, prompting user for debit and credit amounts  
34 for ( size\_t i = 0; i < accounts.size(); i++ )  
35 {  
36 cout << "Account " << i + 1 << " balance: $"  
37 << /\* Call the getBalance function through Account pointer i \*/;  
38  
39 double withdrawalAmount = 0.0;  
40 cout << "\nEnter an amount to withdraw from Account " << i + 1  
41 << ": ";  
42 cin >> withdrawalAmount;  
43 /\* Call the debit function through Account pointer i \*/  
44  
45 double depositAmount = 0.0;  
46 cout << "Enter an amount to deposit into Account " << i + 1  
47 << ": ";  
48 cin >> depositAmount;  
49 /\* Call the credit function through Account pointer i \*/  
50*

*51 // downcast pointer  
52 SavingsAccount \*savingsAccountPtr =  
53 /\* Write a dynamic\_cast operation to to attempt to downcast  
54 Account pointer i to a SavingsAccount pointer \*/  
55  
56 // if Account is a SavingsAccount, calculate and add interest  
57 if ( /\* Write a test to determine if savingsAccountPtr isn't 0 \*/ )  
58 {  
59 double interestEarned = /\* Call member function calculateInterest  
60 through savingsAccountPtr \*/;  
61 cout << "Adding $" << interestEarned << " interest to Account "  
62 << i + 1 << " (a SavingsAccount)" << endl;  
63 /\* Use the credit function to credit interestEarned to  
64 the SavingsAccount pointed to by savingsAccountPtr\*/  
65 } // end if  
66  
67 cout << "Updated Account " << i + 1 << " balance: $"  
68 << /\* Call the getBalance function through Account pointer i \*/  
69 << "\n\n";  
70 } // end for  
71  
72 return 0;  
73 } // end main*

1. **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.