题12.10

**Account Inheritance Hierarchy**

题目要求：

创建一个继承机制的银行账户，以Account为基类，SavingsAccount和CheckingAccount为派生类，基类提供一个double型的数据成员balance表示账户余额，三个成员函数（credit()，debit()，getbalance()）分别用于存款，取款与查询余额。SavingsAccount提供一个double型的数据成员InterestRate表示利率，提供一个calculateInterest()函数用于计算利息。CheckingAccount要求有一个double型的数据成员Checkingfee表示每笔交易的费用，并重写成员函数credit()与debit()，在重写函数中调用基类函数实现存取款，并扣除交易费用。

解题思路：

基类的debit()函数定义为bool型，当成功扣款时返回true，余额不足时返回false并报告错误信息。在CheckingAccount的重写debit()中，首先调用基类的debit()函数，若返回的为true则扣除手续费，若为false则不扣除。

源代码：

//12.10.cpp

# include <iostream>

# include "Account.h"

# include "CheckingAccount.h"

# include "SavingsAccount.h"

using namespace std;

int main()

{

Account num1(2000.0);

SavingsAccount num2(2000.0, 0.03);

CheckingAccount num3(2000.0, 10.0);

cout << "Balance of num1:" << num1.getbalance() << endl;

cout << "Balance of num2:" << num2.getbalance() << endl;

cout << "Balance of num3:" << num3.getbalance() << endl;

cout << "\n";

num1.credit(1000.0);

num2.credit(1000.0);

num3.credit(1000.0);

cout << "Balance of num1:" << num1.getbalance() << endl;

cout << "Balance of num2:" << num2.getbalance() << endl;

cout << "Balance of num3:" << num3.getbalance() << endl;

cout << "\n";

num1.debit(1000.0);

num2.debit(1000.0);

num3.debit(1000.0);

cout << "Balance of num1:" << num1.getbalance() << endl;

cout << "Balance of num2:" << num2.getbalance() << endl;

cout << "Balance of num3:" << num3.getbalance() << endl;

cout << "\n";

cout << "Balance of num2 (before calculateInterest):" << num2.getbalance() << endl;

num2.calculateInterest();

cout << "Balance of num2 (after calculateInterest):" << num2.getbalance() << endl;

cout << "\n";

system("pause");

return 0;

}

//Account.h

# include <iostream>

# ifndef Account\_h

# define Account\_h

using namespace std;

class Account {

public:

Account(double a = 0)

{

if (a >= 0)

balance = a;

else

{

balance = 0;

cout << "Error input !";

}

}

void credit(double a) { balance += a; };

bool debit(double a)

{

if (balance >= a)

{

balance -= a;

return true;

}

else

{

cout << "Debit amount exceeded account balance.";

return false;

}

}

double getbalance() { return balance; };

private:

double balance;

};

#endif

//SavingsAccount.h

# include <iostream>

# include "Account.h"

# ifndef SavingsAccount\_h

# define SavingsAccount\_h

class SavingsAccount : public Account

{

public:

SavingsAccount(double = 0, double = 0);

double calculateInterest();

private:

double interestRate;

};

# endif

//SavingsAccount.cpp

# include <iostream>

# include "SavingsAccount.h"

SavingsAccount::SavingsAccount(double a, double b)

:Account(a)

{

interestRate = b;

}

double SavingsAccount::calculateInterest()

{

double Interest = getbalance()\*interestRate;

credit(Interest);

return Interest;

}

//CheckingAccount.h

# include <iostream>

# include "Account.h"

# ifndef CheckingAccount\_h

# define CheckingAccount\_h

class CheckingAccount :public Account

{

public:

CheckingAccount(double a=0, double b=0);

void credit(double);

void debit(double);

private:

double Checkingfee;

};

# endif

//CheckingAccount.cpp

# include <iostream>

# include "CheckingAccount.h"

using namespace std;

CheckingAccount::CheckingAccount(double a, double b)

:Account(a)

{

Checkingfee = b;

}

void CheckingAccount::credit(double a)

{

Account::credit(a - Checkingfee);

}

void CheckingAccount::debit(double a)

{

if (Account::debit(a))

Account::debit(Checkingfee);

else

cout << "Error ! (Balance is not enough)";

}

运行结果：

