Hassan Harajly

11/21/17

**Bank account program**

1. **Problem Statement**

Create a program that will mimic a banking program in which it has the capability to store data for saving and checking accounts

1. **Requirements**
   1. **Assumptions**

The user will input data via the command line

The user will enter both integers and decimals

* 1. **Specifications**

The program will select multiple accounts, user can use a checking account or a savings account.

The program will have a field for interest rate, cannot be negative. The program will store the interest and balance and account number. Then the interest will be calculated based on years balance and interest rate.

1. **Decomposition Diagram** (Used to break program down into components visually. Diagram can have as many components as needed. Defines functionality that will solve the problem – does NOT define a flow of actions)

Bank program

input

process

output

User will enter data

Program will determine which class during runtime

Program will output data onto the screen

Program will output options for ways to use program

Program will print out error messages

Some Data will be preloaded and preset

Program will load data inside of classes

1. **Test Strategy**

* Valid Data
* Invalid Data

1. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Invalid Data | 1 | Test if balance is negative |  |  |  |  |
| Invalid Data | 2 | Interest rate is<0 |  |  |  |  |
| Valid Data | 3 | Balance=250 |  |  |  |  |

1. **Initial Algorithm**

Class account

Long accNum=0

Double balance=0

Public

Account (){}

Input acc Num

Input Balance

End class

Class checking inherits from account

Int minintrbalance

Input minintrbalnce

Input intrate

End class

Class savings :public account

Double intrate =0

Input intrate

End class

1. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Invalid Data | 1 | Test if balance is negative | -1 | "balance Cannot be negative!" |  |  |
| Invalid Data | 2 | Interest rate is<0 | -1 | "interest rate cannot be negative!" |  |  |
| Valid Data | 3 | Balance=250 | 250 | none |  |  |
| Valid data | 4 | Interest rate>0 | .5 | none |  |  |
| Valid data | 5 | Account number | 123 | none |  |  |
| valid data | 6 | Interest rate | 0 | none |  |  |

1. **Code**

**//hassan harajly**

**//cis200**

**//11/22/17**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class account {**

**private:**

**long accNum = 0;**

**double balance = 0;**

**public:**

**account() {}**

**virtual int setRate(double f) { return 0; }**

**virtual double computeIntr(int years)**

**{**

**return 0;**

**}**

**virtual long getAcct()**

**{**

**return accNum;**

**}**

**double getBalance() {**

**return balance;**

**cout << "inside balance" << balance << " yo ";**

**}**

**int setAcct(long acct)**

**{**

**try {**

**if (acct >= 0)**

**{**

**accNum = +acct;**

**}**

**else**

**{**

**throw "account number cant be negative!";**

**}**

**}**

**catch (string x)**

**{**

**cout << x << endl;**

**return -1;**

**}**

**}**

**int setBalance(double balanc)**

**{**

**try**

**{**

**if (balanc >= 0)**

**{**

**balance = +balanc;**

**}**

**else**

**{**

**throw "balance Cannot be negative!";**

**}**

**}**

**catch (string g)**

**{**

**cout << g << endl;**

**return -1;**

**}**

**return 0;**

**}**

**account(long acct, double bal)**

**{**

**try**

**{**

**if (bal >= 0)**

**{**

**balance = bal;**

**}**

**else**

**{**

**throw "balance Cannot be negative!";**

**}**

**}**

**catch (string g)**

**{**

**cout << g << endl;**

**}**

**try {**

**if (acct >= 0)**

**{**

**accNum = acct;**

**}**

**else**

**{**

**throw "account number cant be negative!";**

**}**

**}**

**catch (string x)**

**{**

**cout << x << endl;**

**}**

**}**

**virtual string toString()**

**{**

**string Balance = to\_string(getBalance());**

**string Acc = to\_string(getAcct());**

**cout << "Account Number " << Acc << ",has a balance of " << Balance;**

**return Balance;**

**}**

**virtual double computeintr(int years)**

**{**

**return 0;**

**}**

**};**

**class checking :public account**

**{**

**private:**

**int minIntrBalance = 0;**

**int intRate = 0;**

**public:**

**checking(){}**

**virtual string toString()**

**{**

**string Balance = to\_string(getBalance());**

**string Acc = to\_string(getAcct());**

**cout << "Account Number " << Acc << ",has a balance of " << Balance;**

**return Balance;**

**}**

**int getMinIntrBalance() { return minIntrBalance; }**

**int setMinIntrBalance(int lowestBalance)**

**{**

**try {**

**if (lowestBalance >= 0)**

**{**

**minIntrBalance = lowestBalance;**

**}**

**else**

**{**

**throw "the minimum balance cant be less than 0!";**

**return -1;**

**}**

**}**

**catch (string k)**

**{**

**cout << k << endl;**

**}**

**}**

**int getRate() { return intRate; }**

**int setRate(double rate)**

**{**

**try {**

**if (rate >= 0)**

**{**

**intRate = rate;**

**}**

**else { throw string("rate cannot be negative!"); }**

**}**

**catch (string k)**

**{**

**cout << k << endl;**

**return -1;**

**}**

**}**

**double computeIntr(int years)**

**{**

**float temp = (float)intRate\*.01;**

**float tempb = getBalance();**

**minIntrBalance = getBalance() + tempb\*temp\*years;**

**return minIntrBalance;**

**}**

**checking(long acc, double bal)**

**{**

**account a(acc, bal);**

**setBalance(bal);**

**setAcct(acc);**

**}**

**};**

**class saving :public account**

**{**

**private:**

**double intRate = 0;**

**public:**

**saving(){}**

**double getRate() { return intRate; }**

**int setRate(double rate) {**

**try {**

**if (rate >= 0)**

**{**

**intRate = rate;**

**}**

**else**

**throw string("interest rate cannot be negative!");**

**}**

**catch (string x)**

**{**

**cout << x << endl;**

**return -1;**

**}**

**}**

**virtual string toString()**

**{**

**string Balance = to\_string(getBalance());**

**string Acc = to\_string(getAcct());**

**cout << "Account Number " << Acc << ",has a balance of " << Balance;**

**return Balance;**

**}**

**double computeIntr(int years)**

**{**

**float minIntrBalance = 0;**

**float temp = (float)intRate\*.01;**

**float tempb = getBalance();**

**minIntrBalance = getBalance() + tempb\*temp\*years;**

**return minIntrBalance;**

**}**

**saving(long acc, double bal)**

**{**

**account a(acc, bal);**

**setBalance(bal);**

**setAcct(acc);**

**}**

**};**

**#define BALANCE 1000**

**int main()**

**{**

**int years = 0;**

**double interestrate = 0;**

**int accountnumber = 0;**

**const int numberofchecking = 5;**

**const int numofsaving = 5;**

**checking c;**

**checking checkingaccount[numberofchecking];**

**saving savings[numofsaving];**

**account\* ac[100];**

**char choice;**

**for (int i = 0; i < numberofchecking; i++)**

**{**

**ac[i] = &checkingaccount[i];**

**ac[i]->setAcct(100 + i);**

**ac[i]->setBalance(BALANCE + (10 \* (ac[i]->getAcct())));**

**}**

**for (int i = 0; i < numofsaving; i++)**

**{**

**ac[numberofchecking + i] = &savings[i];**

**ac[numberofchecking + i]->setAcct(200 + i);**

**ac[numberofchecking + i]->setBalance(BALANCE + (10 \* (ac[numberofchecking + i]->getAcct())));**

**ac[numberofchecking + i]->setRate(.03);**

**}**

**for (int z = 0; z < 1;)**

**{**

**cout << "Welcome to The University of Michigan's Banking Program" << endl;**

**cout << "Enter an account number" << endl;**

**cin >> accountnumber;**

**for (int i = 0; i < 1;)**

**{**

**while (!((accountnumber <= 105 && accountnumber >= 100) || (accountnumber <= 205 && accountnumber >= 200)))**

**{**

**cout << "accountnumber number must be between 100 and 105 or inbetween 200 and 205" << endl;**

**cin >> accountnumber;**

**}**

**if ((accountnumber % 200) >= 0 && accountnumber % 200 <= 5)**

**{**

**cout << "a to add an account\nr to remove account\ng to get account\nb to set balance\ni to set interest\nd to display balance\nc for interest displaying\nq to quit processing this account" << endl;**

**cin >> choice;**

**if (choice == 'r' || choice == 'R')**

**{**

**cout << "account removed!" << endl;**

**i++;**

**}**

**if (choice == 'q' || choice == 'Q')**

**{**

**i++; z++ ;**

**}**

**if (choice == 'g' || choice == 'G')**

**{**

**cout <<"account#: "<< ac[numberofchecking + accountnumber % 200]->getAcct() << endl;**

**}**

**if (choice == '0')**

**{**

**i++;**

**}**

**if (choice == 'd' || choice == 'D')**

**{**

**ac[numberofchecking + accountnumber % 200]->toString();**

**}**

**if (choice == 'i' || choice == 'I')**

**{**

**cout << "What would you like to set the interest rate to?" << endl;**

**cin >> interestrate;**

**ac[numberofchecking + accountnumber % 200]->setRate(interestrate);**

**while (ac[numberofchecking + accountnumber % 200]->setRate(interestrate) == -1)**

**{**

**cout << "Enter again" << endl;**

**cin >> interestrate;**

**}**

**ac[numberofchecking + accountnumber % 200]->setRate(interestrate);**

**}**

**if (choice == 'c' || choice == 'C')**

**{**

**cout << "How many Years would you like to set your interest rate calculation?" << endl;**

**cin >> years;**

**while (years < 0)**

**{**

**cout << "Years cannot be negative! reenter please." << endl;**

**cin >> years;**

**}**

**cout << "your interest rate compounded is: " << ac[numberofchecking + accountnumber % 200]->computeIntr(years);**

**}**

**if (choice == 'b' || choice == 'B')**

**{**

**int bal=0;**

**cout << "please input a new balance" << endl;**

**cin >> bal;**

**while (bal < 0)**

**{**

**cout << "balance Cannot be negative! please reenter" << endl;**

**cin >> bal;**

**}**

**ac[numberofchecking + accountnumber % 200]->setBalance(bal);**

**}**

**if (choice == 'a' || choice == 'A')**

**{**

**int aacc;**

**cout << "enter a new bank account number" << endl;**

**cin >> aacc;**

**cout << "account " << aacc << " has been created" << endl;**

**}**

**}**

**else if ((accountnumber % 100) >= 0 && (accountnumber % 100) <= 5)**

**{**

**//cout <<"Your Balance is: "<< ac[accountnumber % 100]->getBalance();**

**int interestrate = 0;**

**cout << "Your Balance is: " << ac[accountnumber % 100]->getBalance();**

**cout << "What would you like to set the interest rate to?" << endl;**

**cin >> interestrate;**

**ac[accountnumber % 100]->setRate(interestrate);**

**while (ac[accountnumber % 100]->setRate(interestrate) == -1)**

**{**

**cout << "Enter again" << endl;**

**cin >> interestrate;**

**}**

**ac[accountnumber % 100]->setRate(interestrate);**

**int years = 0;**

**cout << "How many Years would you like to set your interest rate calculation?" << endl;**

**cin >> years;**

**while (years < 0)**

**{**

**cout << "Years cannot be negative! reenter please." << endl;**

**cin >> years;**

**}**

**cout << "your interest rate compounded is: " << ac[accountnumber % 100]->computeIntr(years);**

**}**

**}**

**}**

**system("pause");**

**return 0;**

**}**

1. **Updated Algorithm**

accNum = 0;  
balance = 0;  
setRate(f) { return 0; }  
computeIntr(years)  
{  
return 0;  
}  
getAcct()  
{  
return accNum;  
}  
getBalance() {  
return balance;  
}  
setAcct(acct)  
{  
if (acct >= 0)  
{  
accNum = acct;  
}  
otherwise  
{  
print ("account number cant be negative!"};  
}  
catch (string x)  
{  
print(x);  
return -1;  
}  
}  
setBalance(balanc)  
{  
    
if (balanc >= 0)  
{  
balance = +balanc;  
}  
otherwise  
{  
print ("balance cant be negative!"};  
}  
catch (string g)  
{  
print(q);  
return -1;  
}  
return 0;  
}  
account(long acct, double bal)  
{  
if (bal >= 0)  
{  
balance = bal;  
}  
otherwise  
{  
print ("balance cant be negative!"};  
}  
catch (string g)  
{  
cout << g << endl;  
}  
if (acct >= 0)  
{  
accNum = acct;  
}  
else  
{  
print ("account number cant be negative!"};  
}  
catch (string x)  
{  
print(x);  
}  
}  
toString()  
{  
Balance = getBalance();  
Acc = getAcct();  
print(Acc,Balance);  
return Balance;  
}  
double computeintr(years)  
{  
return 0;  
}  
};  
class checking :account  
{  
minIntrBalance = 0;  
intRate = 0;  
checking(){}  
string toString()  
{  
Balance = getBalance();  
Acc = getAcct();  
print(Acc,Balance);  
return Balance;  
}  
getMinIntrBalance() { return minIntrBalance; }  
setMinIntrBalance(int lowestBalance)  
{  
    
if (lowestBalance >= 0)  
{  
minIntrBalance = lowestBalance;  
}  
otherwise  
{  
print("the minimum balance cant be less than 0!");  
return -1;  
}  
catch (string k)  
{  
print(k);  
}  
}  
int getRate() { return intRate; }  
int setRate(double rate)  
{  
if (rate >= 0)  
{  
intRate = rate;  
}  
otherwise { print ("rate cannot be negative!"); }  
    
catch (string k)  
{  
print(k);  
return -1;  
}  
}  
computeIntr(years)  
{  
temp = intRate\*.01;  
tempb = getBalance();  
minIntrBalance = getBalance() + tempb\*temp\*years;  
return minIntrBalance;  
}  
checking(acc, bal)  
{  
account a(acc, bal);  
setBalance(bal);  
setAcct(acc);  
}  
};  
class saving : account  
{  
intRate = 0;  
saving(){}  
getRate() { return intRate; }  
setRate(double rate) {  
if (rate >= 0)  
{  
intRate = rate;  
}  
else  
print("interest rate cannot be negative!");  
catch (string x)  
{  
print x;  
return -1;  
}  
}  
string toString()  
{  
Balance = getBalance();  
Acc = getAcct();  
print(Acc,Balance);  
return Balance;  
}  
computeIntr( years)  
{  
minIntrBalance = 0;  
temp = (intRate\*.01;  
tempb = getBalance();  
minIntrBalance = getBalance() + tempb\*temp\*years;  
return minIntrBalance;  
}  
saving(acc, bal)  
{  
account a(acc, bal);  
setBalance(bal);  
setAcct(acc);  
}  
};  
BALANCE = 1000  
main()  
{  
years = 0;  
interestrate = 0;  
accountnumber = 0;  
numberofchecking = 5;  
numofsaving = 5;  
checking c;  
checking checkingaccount[numberofchecking];  
saving savings[numofsaving];  
account\* ac[100];  
for i = 0 to numberofchecking  
{  
ac[i] = &checkingaccount[i];  
ac[i]->setAcct(100 + i);  
ac[i]->setBalance(BALANCE + (10 \* (ac[i]->getAcct())));  
}  
for i = 0 to numberofsaving  
{  
ac[numberofchecking + i] = &savings[i];  
ac[numberofchecking + i]->setAcct(200 + i);  
ac[numberofchecking + i]->setBalance(BALANCE + (10 \* (ac[numberofchecking + i]->getAcct())));  
ac[numberofchecking + i]->setRate(.03);  
}  
for z=0 to z=1 without increment  
{  
print( "Welcome to The University of Michigan's Banking Program");  
input( Valid account number);  
if ((accountnumber % 200) >= 0 && accountnumber % 200 <= 5)  
{  
print ("a to add an account ,r to remove account, g to get account,b to set balance,i to set interest,d to display balance,c for interest displaying,q to quit processing this account");  
input( choice );  
if (choice == 'r' Or choice == 'R')  
{  
print (account\_removed)  
}  
if (choice == 'q' OR choice == 'Q')  
{  
z=1;  
}  
if (choice == 'g' OR choice == 'G')  
{  
print(accountnumber);  
}

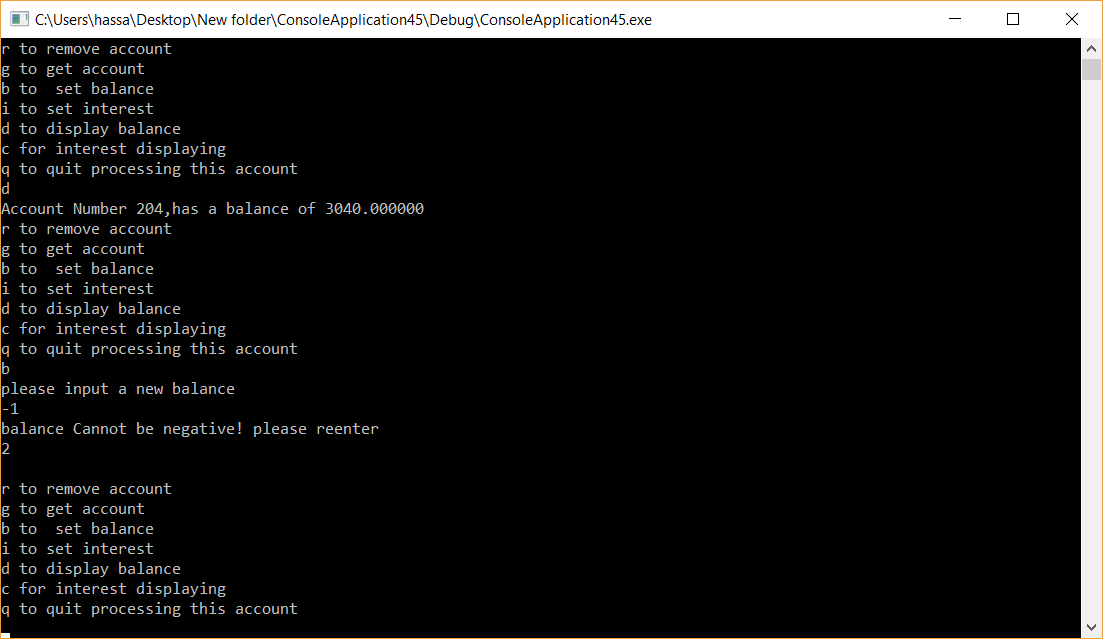
if (choice == 'd' OR choice == 'D')  
{  
print(balance);  
}  
if (choice == 'i' OR choice == 'I')  
{  
print( "What would you like to set the interest rate to?");  
input( correct interestrate);  
ac[numberofchecking + accountnumber % 200]->setRate(interestrate);  
if (choice == 'c' OR choice == 'C')  
{  
print( "How many Years would you like to set your interest rate calculation?" );  
input ( years );  
    
print( Compounded interest rate ) }  
if (choice == 'b' OR choice == 'B')  
{  
int bal=0;  
print("please input a new balance" )  
input(bal);  
ac[numberofchecking + accountnumber % 200]->setBalance(bal);  
}  
if (choice == 'a' OR choice == 'A')  
{  
int aacc;  
print( "enter a new bank account number" );  
cin >> aacc;  
print("acc has been created");  
}  
}  
else if ((accountnumber % 100) >= 0 && (accountnumber % 100) <= 5)  
{  
int interestrate = 0;  
print ( Balance );  
print("What would you like to set the interest rate to?");  
input(interestrate);  
ac[accountnumber % 100]->setRate(interestrate);  
int years = 0;  
print("How many Years would you like to set your interest rate calculation?");  
input(years);  
print( compounded interestrate);  
}  
    
}  
pause\_system;  
return 0;  
}

1. **Test Plan Version 3**

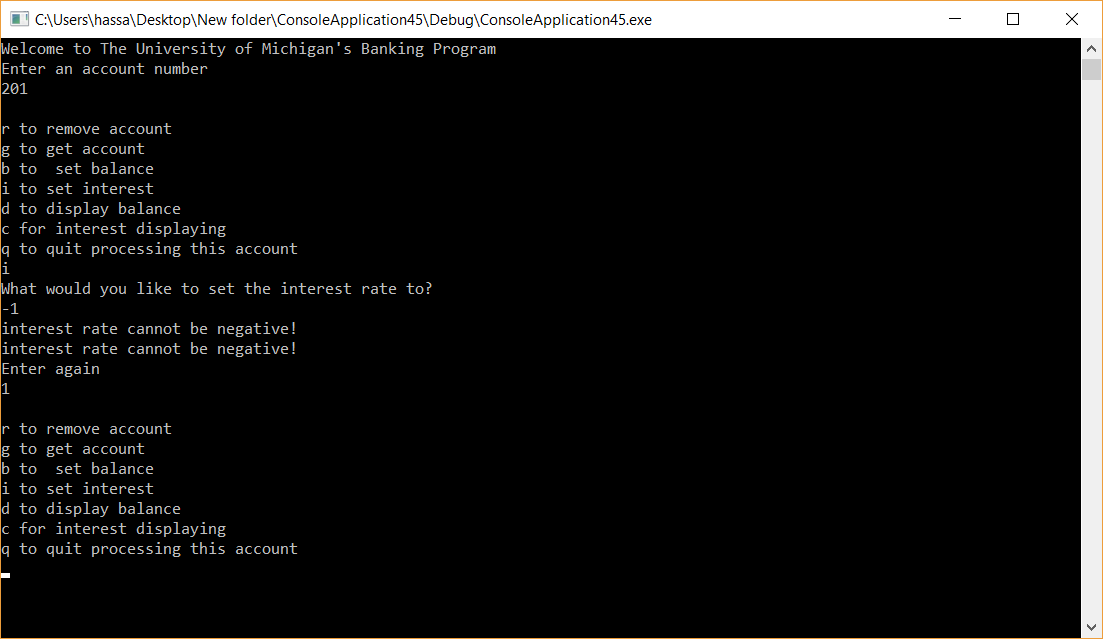
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Invalid Data | 1 | Test if balance is negative | -1 | "balance Cannot be negative!" | "balance Cannot be negative! Please reenter" | pass |
| Invalid Data | 2 | Interest rate is<0 | -1 | "interest rate cannot be negative!" | "interest rate cannot be negative!" | pass |
| Valid Data | 3 | Balance=250 | 250 | none | None | pass |
| Valid data | 4 | Interest rate>0 | .5 | none | None | pass |
| Valid data | 5 | Account number | 123 | none | Account 123 has been created | Pass |
| valid data | 6 | Interest rate | 0 | none | None | pass |
| Valid data | 7 | Add account | A,209 | Account 209 created | Account 209 created | pass |

1. **Screenshots**

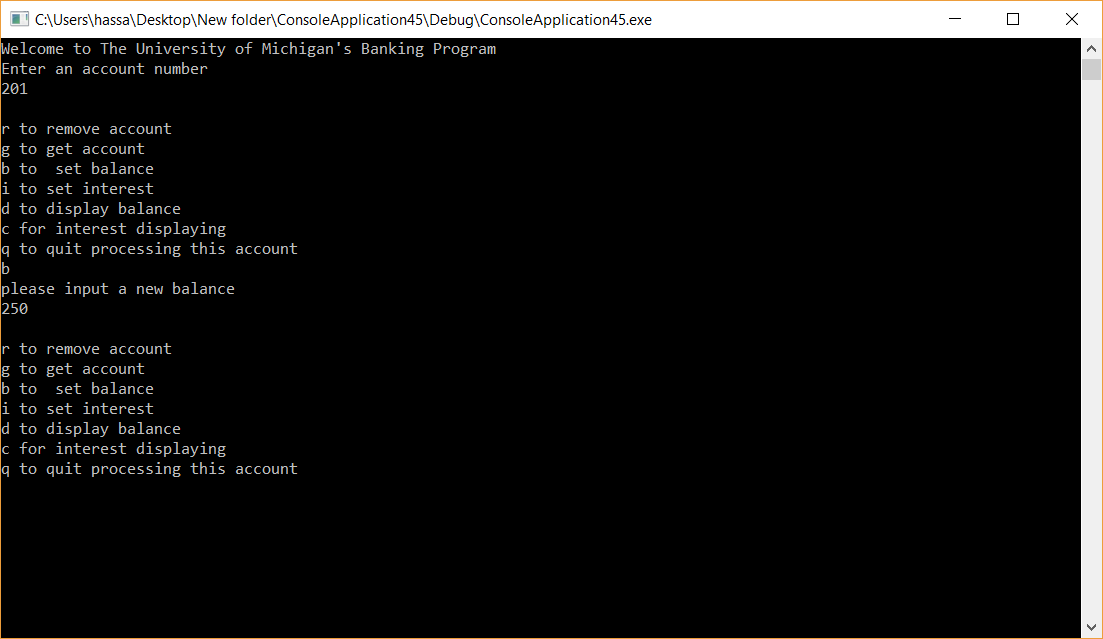
Test Cases 1-9

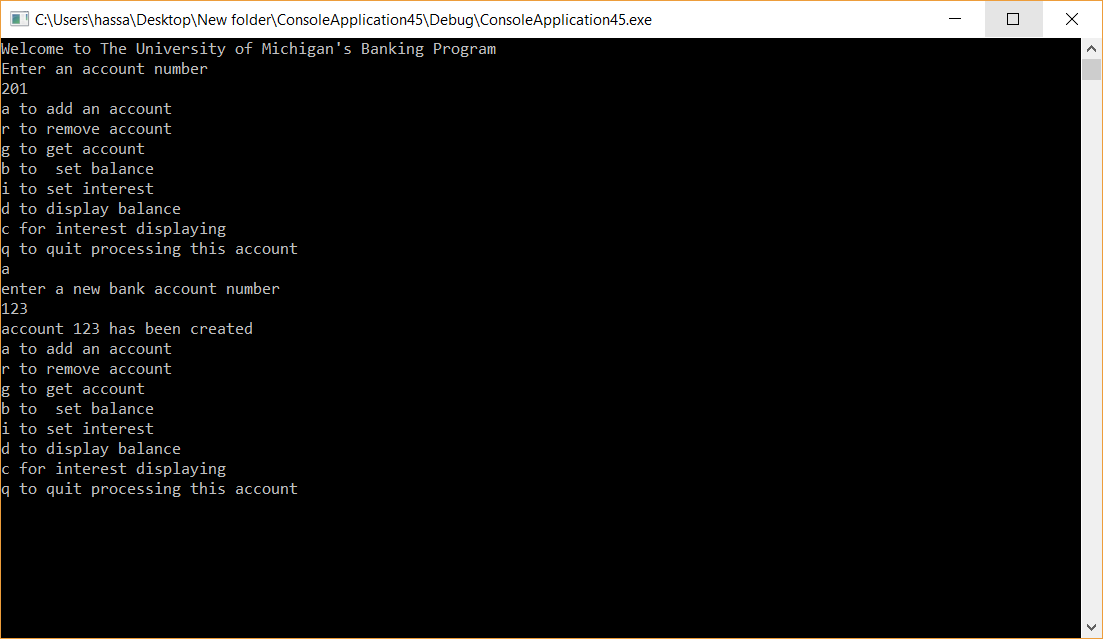


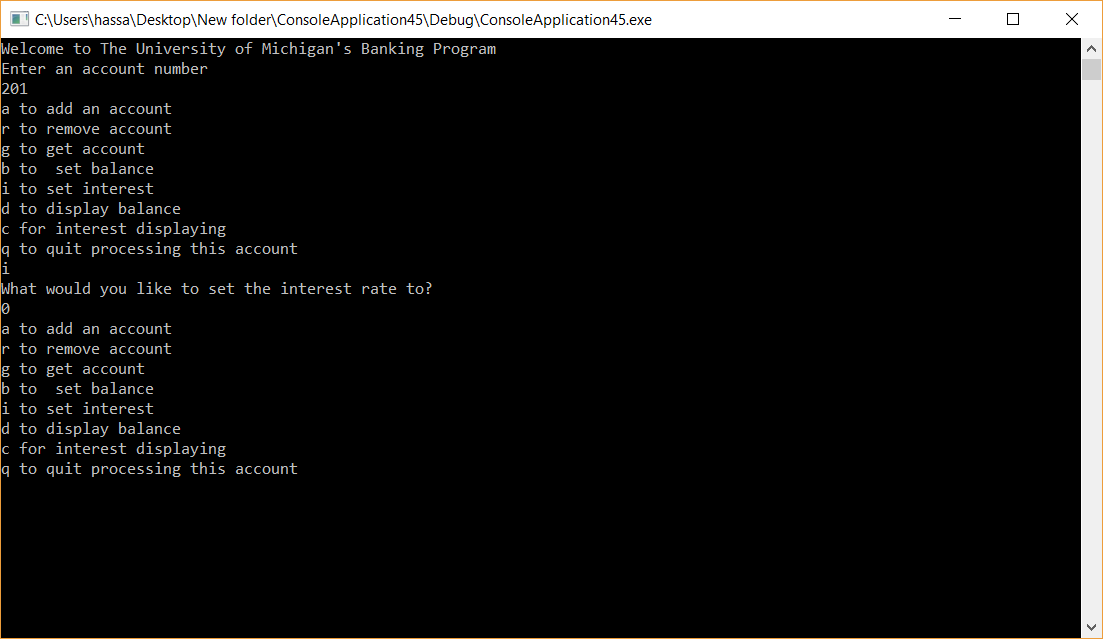
**2**

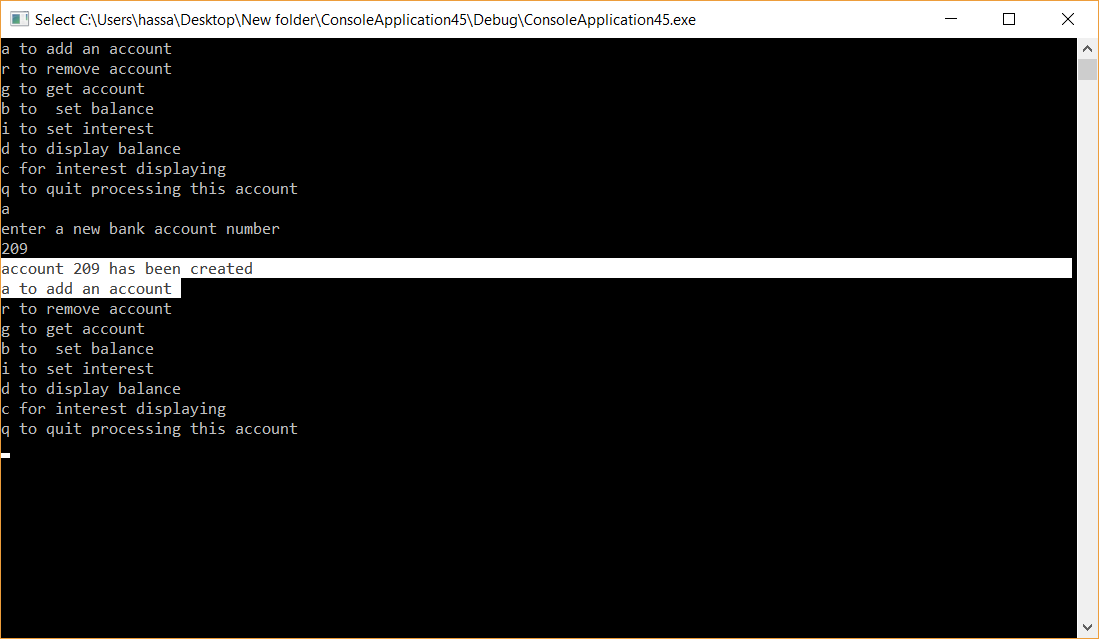


**3**









1. **Error Log**

|  |  |  |
| --- | --- | --- |
| Error Type | Cause of Error | Solution to Error |
|  |  |  |

1. **Status**

The program works 100% with assumptions in place