**Program Description**

Bank account class with derived checking and savings account classes.

**Program Execution**

A computer screen capture

Description automatically generated with medium confidence

**Reflection**

I used the code from the first week to construct the week2 assignment. A big bug difficult to find was properly declaring the parameterized constructor in the derived class headers. I googled for examples and fixed the error.

**Program Code** //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*bankAccountTest.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#ifndef DC\_H

#define DC\_H

#include <iostream>

#include <iomanip>

#include "savingsAccount.h"

#include "checkingAccount.h"

#include "bankAccount.h"

using namespace std;

int main()

{

int accountNumber = 1000;

checkingAccount jackAccount(accountNumber++,1000);

checkingAccount lisaAccount(accountNumber++, 450);

savingsAccount samirAccount(accountNumber++, 9300);

savingsAccount ritaAccount(accountNumber++, 32);

jackAccount.deposit(1000);

lisaAccount.deposit(2300);

samirAccount.deposit(800);

ritaAccount.deposit(500);

jackAccount.postInterest();

lisaAccount.postInterest();

samirAccount.postInterest();

ritaAccount.postInterest();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

jackAccount.print();

lisaAccount.print();

samirAccount.print();

ritaAccount.print();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

jackAccount.writeCheck(250);

lisaAccount.writeCheck(350);

samirAccount.withdraw(120);

ritaAccount.withdraw(290);

cout << "\*\*\*\*\*\*\*\*After withdrawals \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

jackAccount.print();

lisaAccount.print();

samirAccount.print();

ritaAccount.print();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

return 0;

}

#endif // !DC\_H

//BASE CLASS DEFINITION

// bankAccount base class

#pragma once

#ifndef BANKACCOUNT\_H

#define BANKAACOUNT\_H

#include <string>

#include <stdlib.h>

#include <iostream>

using namespace std;

class bankAccount

{

public:

//DEFAULT CONSTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bankAccount();

//PARAMETERIZED CONSTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bankAccount(int, double);

//CONSTANT MEMBER FUNCTIONS

int getAccountNumber() const;

double getBalance() const;

void print() const;

//MODIFICATION MEMBER FUNCTIONS

void setAccountNumber(int userAccountNumber);

void deposit(double moneyIn);

void withdraw(double moneyOut);

protected:

double balance;

int accountNumber;

};

#endif

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*IMPLEMENTATION FILE BASE BANKACCOUNT

#pragma once

#ifndef BA\_H

#define BA\_H

#include "bankAccount.h"

//#include <string>

//#include <iostream>

#include <iomanip>

//DEFAULT CONSTRUCTOR

bankAccount::bankAccount()

{

accountNumber = 0;

balance = 0.0;

};

//PARAMETERIZED CONSTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bankAccount::bankAccount(int userAccountNumber, double userAccountBalance)

{

accountNumber = userAccountNumber;

balance = userAccountBalance;

};

//MODIFICATION MEMBER FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void bankAccount::deposit(double moneyIn)

{

balance += moneyIn;

};

void bankAccount::withdraw(double moneyOut)

{

balance -= moneyOut;

};

void bankAccount::setAccountNumber(int userAccountNumber)

{

accountNumber = userAccountNumber;

};

//CONSTANT MEMBER FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int bankAccount::getAccountNumber() const

{

return accountNumber;

};

double bankAccount::getBalance() const

{

return balance;

};

void bankAccount::print() const

{

cout << fixed << showpoint;

cout << setprecision(2)

<< " Account number: " << accountNumber

<< " Account Balance: " << balance

<< endl << endl << endl;

};

#endif // !BA\_H

//CHECKING ACCOUNT HEADER FILE

#pragma once

#ifndef CHECKING\_H

#define CHECKING\_H

//#include <string>

//#include <stdlib.h>

#include "bankAccount.h"

//using namespace std;

//DERIVED CLASS checkingAccout

class checkingAccount : public bankAccount

{

public:

//DEFAULT CONSTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// checkingAccount();

//PARAMETERIZED CONSTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

checkingAccount(int, double);

//MODIFICATION MEMBER FUNCTIONS

void setMinimumBalance(double);

void setInterestRate(double);

void setServiceCharge(double);

void withdraw(double);

void writeCheck(double);

void postInterest();

void deposit(double);

//CONSTANT MEMBER FUNCTIONS

double getMinimumBalance(int) const;

double getInterestRate(int) const;

double getServiceCharge(int) const;

void print() const;

private:

double interestRate = 0;

double minimumBalance = 0;

double serviceCharge =0;

};

#endif

//IMPLEMENTATION FILE CHECKING BANKACCOUNT

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef CA\_H

#define CA\_H

#include "checkingAccount.h"

#include <string>

#include <iostream>

#include <iomanip>

//DEFAULT CONTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

checkingAccount::checkingAccount()

{

interestRate = 0;

minimumBalance = 0;

serviceCharge = 0;

}

\*/

//PARAMETERIZED CONSTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

checkingAccount::checkingAccount(int userAccountNumber, double userBalance) : bankAccount(userAccountNumber, userBalance)

{

accountNumber = userAccountNumber;

balance = userBalance;

minimumBalance = 0;

interestRate = 0.0;

serviceCharge = 0.0;

};

////MODIFICATION MEMBER FUNCTIONS

void checkingAccount::setMinimumBalance(double userMinimumBalance)

{

checkingAccount::minimumBalance = userMinimumBalance;

};

void checkingAccount::setInterestRate(double userInterestRate)

{

interestRate = userInterestRate;

};

void checkingAccount::setServiceCharge(double userSetServiceCharge)

{

serviceCharge = userSetServiceCharge;

};

void checkingAccount::withdraw(double withdrawAmount)

{

//withdraw checks to make sure there is enough money in the account first and reports a warning if not.

if (( balance - withdrawAmount) < 0)

{

cout << "There is not enough money in the checking account." << endl;

return;

}

else

{

//If there is enough money to make the withdraw, it then checks to see if the balance will go below the minimum amount after the withdraw.

if ( (balance - withdrawAmount) < checkingAccount::minimumBalance)

{

cout << "The withdrawal will go below the minimum balance allowed." << endl;

//If so, it checks to see if the balance will go below zero after the withdraw and the service charge is applied.

//Should this occur an error is reported.

if ((balance - withdrawAmount) < 0)

{

cout << "There is not enough money in the checking account." << endl;

cout << "A service charge will be applied to the checking account." << endl;

balance -= serviceCharge;

return;

}

else

//If not, the balance is adjusted to reflect the withdraw and the service charge applied.

{

balance -= withdrawAmount;

balance -= serviceCharge;

return;

}

}

}

//Fallthrough to withdraw from balance

{

balance -= withdrawAmount;

}

};

void checkingAccount::writeCheck(double checkAmount)

//writeCheck calls withdraw

{

checkingAccount::withdraw(checkAmount);

};

void checkingAccount::postInterest()

//postInterest takes the interest amount based off of the total balance and adds it to the balance (balance + balance \* interestRate)

{

balance += (balance \* interestRate);

};

void checkingAccount::deposit(double moneyIn)

{

balance += moneyIn;

};

//CONSTANT MEMBER FUNCTIONS

double checkingAccount::getMinimumBalance(int userAccountNumber) const

{

return minimumBalance;

};

double checkingAccount::getInterestRate(int userAccountNumber) const

{

return interestRate;

};

double checkingAccount::getServiceCharge(int userAccountNumber) const

{

return serviceCharge;

};

void checkingAccount::print() const

{

cout << fixed << showpoint;

cout << setprecision(2)

<< " Account number: " << accountNumber << endl

<< " Account type: Checking" << endl

<< " Balance: " << balance << endl

<< " Interest rate: " << interestRate << endl

<< " Minimum Account Balance: " << minimumBalance << endl

<< " Service charge: " << serviceCharge << endl <<endl << endl;

};

#endif // !CA\_H

/SAVINGS ACCOUNT HEADER FILE

#pragma once

#ifndef SAVINGS\_H

#define SAVINGS\_H

#include <string>

#include <stdlib.h>

#include "bankAccount.h"

//using namespace std;

//DERIVED CLASS savingsAccount

class savingsAccount : public bankAccount

{

public:

//DEFAULT CONSTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

savingsAccount();

//PARAMETERIZED CONSTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

savingsAccount(int, double);

//MODIFICATION MEMBER FUNCTIONS

void setInterestRate(double);

void withdraw(double);

void deposit(double);

//CONSTANT MEMBER FUNCTIONS

double getInterestRate() const;

void postInterest();

void print() const;

private:

double interestRate = 0;

};

#endif

//IMPLEMENTATION FILE SAVINGS BANKACCOUNT

#ifndef SA\_H

#define SA\_H

#include "bankAccount.h"

#include "savingsAccount.h"

#include <string>

#include <iostream>

#include <iomanip>

//DEFAULT CONTRUCTOR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

savingsAccount::savingsAccount()

{

interestRate = 0;

}

//PARAMETERIZED CONSTRUCTOR

savingsAccount::savingsAccount(int userAccountNumber, double userBalance) :bankAccount(userAccountNumber, userBalance)

{

accountNumber = userAccountNumber;

balance = userBalance;

interestRate = 0;

};

//MODIFICATION MEMBER FUNCTIONS

void savingsAccount::setInterestRate(double userInterestRate)

{

interestRate = userInterestRate;

};

void savingsAccount::withdraw(double withdrawAmount)

//Withdraw checks to make sure there is enough money before altering the balance.

{

//withdraw checks to make sure there is enough money in the account first and reports a warning if not.

if ((balance - withdrawAmount) < 0)

{

cout << "There is not enough money in the checking account." << endl;

}

else

//If not, a warning message is printed and the balance remains unchanged.

{

balance -= withdrawAmount;

}

}

void savingsAccount::deposit(double moneyIn)

{

balance += moneyIn;

};

//CONSTANT MEMBER FUNCTIONS

double savingsAccount::getInterestRate() const

{

return interestRate;

}

void savingsAccount::postInterest()

//postInterest takes the interest amount based off of the total balance and adds it to the balance (balance + balance \* interestRate)

{

balance += balance \* interestRate;

}

void savingsAccount::print() const

{

cout << fixed << showpoint;

cout << setprecision(2)

<< " Account number: " << accountNumber << endl

<< " Interest rate: " << interestRate << endl

<< " Account Balance: " << balance << endl << endl << endl;

};

#endif // !SA\_H