

The way the program works is the Statistics lives on top of everything keeping track of what operations are taking place at the bank.

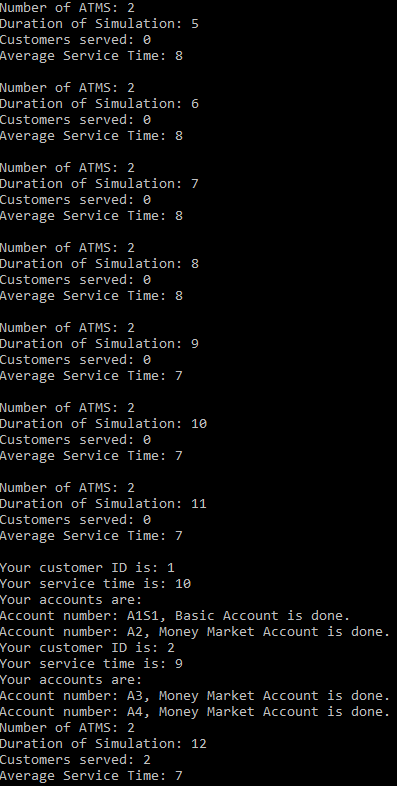
The Traffic Generator lives under the statistics keeper and is utilized to send customers to the bank

The Timing wheel updates the bank when ever a ATM is available to be utilized

The ATMs process customer transactions when they arrive at the ATM

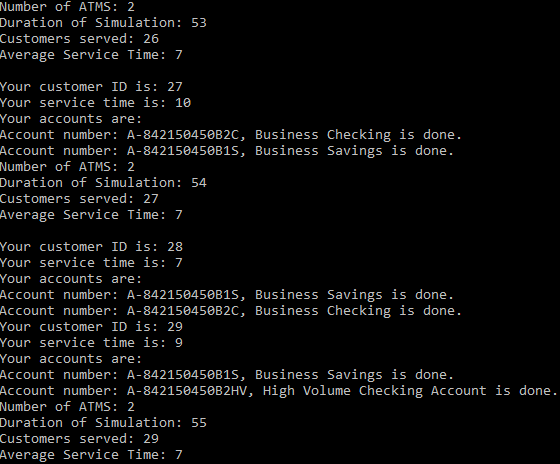
Customers use the network to deposit or withdraw money

Accounts contain customer financial data

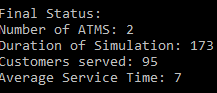


Customers begin arriving at the bank and the statistics keeper begins reporting the ATM status.

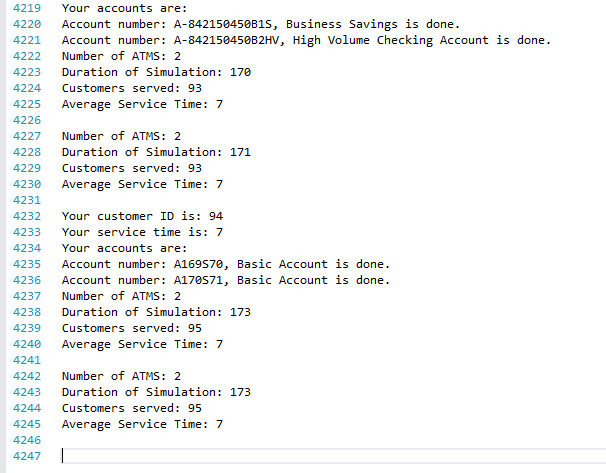
Soon the first customer finishes using the ATM and reports the account number, account type and their service time at the bank.



During each time sequence the status is updated until finally the simulation time is reached and the final statistics report is generated



The final log is found in the output.txt file



Code

#pragma once

#include<iostream>

#include <string>

#include <set>

#include <ostream>

using namespace std;

//Please remember that other than implementing the features mentioned above your hierarchy

//should be able to generate the account ids as detailed in the project description document.

//You can consider the following to be the basic amount of information for an account.You will be

//adding on attributes as needed for the subtypes.

//Account

//Account ID

//SSN

//Nameand Address(optional)

//Balance

class Account

{

protected:

int account\_count;

string accountCountString;

string accountID;

float balance;

float interestRate;

float interestGained;

//string accountID;

string SSN;

string name;

string address;

string accountType;

//static int void accountCount = 0;

public:

Account();

explicit Account(float balance, float interestRate, string SSN);

void virtual AddInterest();

void virtual Transfer(float amount, Account accountID);

void virtual Deposit(float amount , string accountID);

void virtual Withdraw(float amount, string accountID);

void setName(std::string name);

void setAddress(std::string address);

void virtual display();

friend std::ostream& operator<<(std::ostream& os, Account& account);

//void setAccountId();

void reportDone();

};

#include "Account.h"

#include <iostream>

#include <fstream>

using namespace std;

Account::Account()

{

static int account\_count;

account\_count++;

this->accountCountString = to\_string(account\_count);

this->accountID = "A" + accountCountString;

this->balance = 0;

this->interestRate = 1.1;

this->SSN = "000";

this->accountType = "Basic Account";

}

Account::Account(float balance, float interestRate, string SSN)

{

static int account\_count;

account\_count++;

accountCountString = to\_string(account\_count);

cout << "Your account count string : " << accountCountString << endl;

accountID = "A" + accountCountString;

this->balance = balance;

this->interestRate = interestRate;

this->SSN = SSN;

}

void Account::AddInterest()

{

this->interestGained = this->interestGained + (this->balance \* (this->interestRate - 1));

this->balance = this->balance \* this->interestRate;

cout << "Adding interest " << this->interestRate << endl;

cout << "Your new balance is " << this->balance << endl << endl;

}

void Account::Transfer(float amount, Account accountID)

{

this->balance = this->balance - amount;

accountID.balance = accountID.balance + amount;

cout << "Transfering " << amount << "From " << this->accountID << " to " << accountID.accountID<< endl;

}

void Account::Deposit(float amount, string accountID)

{

cout << "Adding " << amount << " to " << accountID << endl;

this->balance = this->balance + amount;

cout << "Your new balance is " << this->balance << endl;

}

void Account::Withdraw(float amount, string accountID)

{

cout << "Withdrawing " << amount << " from " << accountID << endl;

this->balance = this->balance - amount;

cout << "Your new balance is " << this->balance << endl;

}

void Account::setName(std::string name)

{

this->name = name;

}

void Account::setAddress(std::string address)

{

this->address = address;

}

void Account::display()

{

//cout << "This is a regular account" << endl;

cout << "Account number: " << accountID << endl;

std::cout << "Balance: " << balance << endl;

cout << "Interest Rate: " << interestRate << endl;

cout << "SSN: " << SSN << endl << endl;

}

void Account::reportDone()

{

fstream myfile;

myfile.open("output.txt", fstream::app);

myfile << "Account number: " << this->accountID << ", " << this->accountType << " is done. " << endl;

myfile.close();

cout << "Account number: " << this->accountID << ", " << this->accountType << " is done. " << endl;

}

std::ostream& operator<<(std::ostream& os, Account& account)

{

//float balance;

//float interestRate;

//std::string accountID;

//std::string SSN;

//std::string name;

//std::string address;

std::cout << "Balance: " << account.balance << endl;

cout << "Interest Rate: " << account.interestRate << endl;

cout << "Account number: " << account.accountID << endl;

cout << "SSN: " << account.SSN << endl << endl;

return cout;

}

#include "BusinessChecking.h"

BusinessChecking::BusinessChecking()

{

b\_checking\_acc\_count++;

string b\_s\_a\_c = to\_string(b\_checking\_acc\_count);

account\_count++;

string a\_c = to\_string(account\_count);

this->accountID = 'A' + a\_c + 'B' + b\_s\_a\_c + 'C';

this->accountType = "Business Checking";

}

void BusinessChecking::display()

{

cout << "This is Business Checking Account" << endl;

cout << "Account number: " << accountID << endl;

std::cout << "Balance: " << balance << endl;

cout << "SSN: " << SSN << endl << endl;

Deposit(100000, accountID);

Deposit(1, accountID);

Withdraw(99900, accountID);

Withdraw(90, accountID);

cout << endl;

}

void BusinessChecking::Withdraw(float amount, string accountID)

{

freeWithdrawalsLeft--;

if (freeWithdrawalsLeft >= 0)

{

cout << "Withdrawing " << amount << " from " << accountID << endl;

cout << "Free Withdrawals left = " << freeWithdrawalsLeft << endl;

this->balance = this->balance - amount;

cout << "Your new balance is " << this->balance << endl;

}

else

{

cout << "Withdrawing " << amount << " from " << accountID << endl;

cout << "You will be charged for this transaction : " << withdrawalFee << endl;

cout << "Withdrawing " << amount << "+" << withdrawalFee << " from " << accountID << endl;

this->balance = this->balance - amount - withdrawalFee;

cout << "Your new balance is " << this->balance << endl;

}

}

void BusinessChecking::Deposit(float amount, string accountID)

{

freeDeposit--;

if (freeDeposit >= 0)

{

cout << "Depositing " << amount << " to " << accountID << endl;

cout << "Free Deposits left = " << freeDeposit << endl;

this->balance = this->balance + amount;

cout << "Your new balance is " << this->balance << endl;

}

else

{

cout << "Depositing " << amount << " to " << accountID << endl;

cout << "You will be charged for this transaction : " << DepositFee << endl;

cout << "Depositing " << amount << "-" << DepositFee << " to " << accountID << endl;

this->balance = this->balance + amount - DepositFee;

cout << "Your new balance is " << this->balance << endl;

}

}

#pragma once

#include "CheckingAccount.h"

//Business Account : Bank accounts for a business may vary depending on the type of

//business.

//a.Checking Account : Details same as personal checking account but limit on

//number of transactions

class BusinessChecking: public CheckingAccount

{

private:

int transactionsLeft;

int freeWithdrawalsLeft = 1;

int freeDeposit = 1;

float DepositFee = 1.0;

float withdrawalFee = 2.5;

int b\_checking\_acc\_count = 1;

public:

BusinessChecking();

void Deposit(float amount, string accountID);

void Withdraw(float amount, string accountID);

void display();

};

#pragma once

#include "SavingsAccount.h"

//Savings Account : Similar to the aforementioned business accounts in most

//respects, but it also has a rate of interest associated with the monies in the

//account.The numbers of deposits are unlimitedand free but the number of free

//withdrawals is limited.

class BusinessSavings :public SavingsAccount

{

int freeWithdrawalsLeft =1;

float withdrawalFee = 2.5;

int b\_savings\_acc\_count = 0;

public:

BusinessSavings();

void AddInterest();

void Withdraw(float amount, string accountID);

void display();

};

#include "BusinessSavings.h"

BusinessSavings::BusinessSavings()

{

b\_savings\_acc\_count++;

string b\_s\_a\_c = to\_string(b\_savings\_acc\_count);

account\_count++;

string a\_c = to\_string(account\_count);

this->accountID = 'A' + a\_c + 'B' + b\_s\_a\_c + 'S';

this->AddInterest();

this->accountType = "Business Savings";

}

void BusinessSavings::AddInterest()

{

if (balance > 10000)

{

interestRate = 11.0;

}

else

{

interestRate = 8.5;

}

}

void BusinessSavings::display()

{

cout << "This is Business Savings Acccount" << endl;

cout << "Account number: " << accountID << endl;

std::cout << "Balance: " << balance << endl;

cout << "Interest Rate: " << interestRate << endl;

cout << "SSN: " << SSN << endl << endl;

Deposit(100000,accountID);

AddInterest();

cout << "New Interest Rate " << interestRate << endl;

Withdraw(99900, accountID);

Withdraw(90, accountID);

cout << endl;

}

void BusinessSavings::Withdraw(float amount, string accountID)

{

freeWithdrawalsLeft--;

if (freeWithdrawalsLeft >=0)

{

cout << "Withdrawing " << amount << " from " << accountID << endl;

cout << "Free Withdrawals left = " << freeWithdrawalsLeft << endl;

this->balance = this->balance - amount;

cout << "Your new balance is " << this->balance << endl;

}

else

{

cout << "Withdrawing " << amount << " from " << accountID << endl;

cout << "You will be charged for this transaction : " << withdrawalFee<<endl;

cout << "Withdrawing " << amount <<"+"<< withdrawalFee << " from " << accountID << endl;

this->balance = this->balance - amount - withdrawalFee;

cout << "Your new balance is " << this->balance << endl;

}

}#pragma once

#include "Account.h"

//Checking Account

//• Allows holder to make deposits& withdrawals

//• Unlimited number of withdrawals& deposits can be made

//• No interest on the money.

//• Can transfer money to other accounts.

class CheckingAccount: public Account

{

private:

int checking\_acc\_count = 1;

//float interest = 0;

//float balance;

public:

CheckingAccount();

CheckingAccount(float balance, std::string SSN);

void display();

};

#include "CheckingAccount.h"

CheckingAccount::CheckingAccount(float balance, std::string SSN) : Account(balance, 1, SSN)

{

checking\_acc\_count++;

string s\_a\_c = to\_string(checking\_acc\_count);

this->accountID = accountID + 'P' + s\_a\_c + 'S';

this->accountType = "Personal Checking Account";

}

CheckingAccount::CheckingAccount()

{

checking\_acc\_count++;

string s\_a\_c = to\_string(checking\_acc\_count);

checking\_acc\_count++;

string a\_c = to\_string(account\_count);

this->accountID = 'A' + a\_c + 'P' + s\_a\_c + 'S';

SSN = "12345";

}

void CheckingAccount::display()

{

cout << "This is Personal Checking Account" << endl;

cout << "Account number: " << accountID << endl;

std::cout << "Balance: " << balance << endl;

cout << "SSN: " << SSN << endl << endl;

}

#pragma once

//Certificates of Deposits

//• Require the account holder to make a deposit and agree to leave funds

//in the account for a specific amount of time

//• Interests are paid in return of the above - mentioned agreement.

//• Interests paid are higher than any of the afore - mentioned accounts.

//• Withdrawals may be allowed on the interest but only a limited number.

#include "Account.h"

class CODAccount:public Account

{

private:

string codAccountString;

int depositTime;

int interestGained;

int withdrawalsAllowed;

//float defaultInterest = 1.10

//float balance;

//std::set<std::string> documentContents;

//std::set<std::string> search;

//int searchLength;

//int documentLength;

//double simScore;

public:

CODAccount();

void Withdraw(float amount);

//void setWithdrawalsAllowed(int sumOfInterestGained);

void display();

//void AddInterest(float amount);

//void printWords(std::string newString);

//void addWords(std::string newString, std::set<std::string>& whatToAddTo);

//double ComputeSim();

//auto GetDocument() { return documentContents; }

//auto GetSimScore() { return simScore; }

};

#include "CODAccount.h"

//Certificates of Deposits

//• Require the account holder to make a depositand agree to leave funds

//in the account for a specific amount of time

//• Interests are paid in return of the above - mentioned agreement.

//• Interests paid are higher than any of the afore - mentioned accounts.

//• Withdrawals may be allowed on the interest but only a limited number.

CODAccount::CODAccount()

{

this->interestRate = interestRate + .1;

static int codAccountNumber;

codAccountNumber++;

codAccountString = "D" + to\_string(codAccountNumber);

this->accountID = this->accountID + codAccountString;

this->withdrawalsAllowed = 1;

this->depositTime = 12;

this->interestGained = 0;

this->accountType = "COD Account";

}

void CODAccount::Withdraw(float amount)

{

if (this->withdrawalsAllowed > 0)

{

if (amount < this->interestGained)

{

this->interestGained = this->interestGained - amount;

this->balance = this->balance - amount;

this->withdrawalsAllowed--;

cout << "You have withdrawn : " << amount << "Your balance is : " << this->balance << endl;

}

else

{

cout << "Your amount : " << amount << " is too large it needs to be less then : " << this->interestGained << endl;

}

}

else

{

cout << "I'm sorry your withdrawal limit has been reached" << endl;

}

}

void CODAccount::display()

{

cout << "COD Account :" << endl;

cout << "Your interest rate is : " << interestRate << endl;

cout << "Account Number : " << this->accountID << endl;

}

#pragma once

#include "CheckingAccount.h"

#include <iostream>

#include <array>

//Foreign Currency Accounts : Same as Checking Account but would let the user

//deposit in a currency other than dollars.You can choose five world currencies of

//your choiceand use the current exchange rates to implement this feature.

class ForeignCurrencyAccount: public CheckingAccount

{

private:

double exchangeRate[5] = {.85, 6.65, .77, 20.98, 1.31 };

//float balanceInDollars = exchangeRate[0]

public:

enum currency { Euro = 0, Yuan = 1, Pound = 2, Peso = 3, CanadianDollar = 4 };

ForeignCurrencyAccount();

void display();

float toDollars(float amount);

float fromDollars(float amount);

};

#pragma once

#include "CheckingAccount.h"

#include <iostream>

#include <array>

//Foreign Currency Accounts : Same as Checking Account but would let the user

//deposit in a currency other than dollars.You can choose five world currencies of

//your choiceand use the current exchange rates to implement this feature.

class ForeignCurrencyAccount: public CheckingAccount

{

private:

double exchangeRate[5] = {.85, 6.65, .77, 20.98, 1.31 };

//float balanceInDollars = exchangeRate[0]

public:

enum currency { Euro = 0, Yuan = 1, Pound = 2, Peso = 3, CanadianDollar = 4 };

ForeignCurrencyAccount();

void display();

float toDollars(float amount);

float fromDollars(float amount);

};

#include "ForeignCurrencyAccount.h"

ForeignCurrencyAccount::ForeignCurrencyAccount()

{

this->accountType = "Foreign Currency Account";

}

void ForeignCurrencyAccount::display()

{

cout << "This is Foreign Currency Account" << endl;

}

#pragma once

#include "CheckingAccount.h"

//High Volume Checking Account : Similar to Checking account but unlimited

//number of transactions are allowed.However, there is a small transaction fee

//for every deposit or check.

class HighVolumeCheckingAccount: public CheckingAccount

{

private:

float DepositFee = 1.0;

int hvc\_checking\_acc\_count = 1;

public:

HighVolumeCheckingAccount();

void Deposit(float amount, string accountID);

void Withdraw(float amount, string accountID);

void display();

};

#include "HighVolumeCheckingAccount.h"

HighVolumeCheckingAccount::HighVolumeCheckingAccount()

{

hvc\_checking\_acc\_count++;

string hvc\_s\_a\_c = to\_string(hvc\_checking\_acc\_count);

account\_count++;

string a\_c = to\_string(account\_count);

this->accountID = 'A' + a\_c + 'B' + hvc\_s\_a\_c + "HV";

this->accountType = "High Volume Checking Account";

}

void HighVolumeCheckingAccount::display()

{

cout << "This is High Volume Checking Account" << endl;

cout << "Account number: " << accountID << endl;

std::cout << "Balance: " << balance << endl;

cout << "SSN: " << SSN << endl << endl;

Deposit(100, accountID);

Deposit(10, accountID);

Withdraw(1, accountID);

Withdraw(1, accountID);

cout << endl;

}

void HighVolumeCheckingAccount::Withdraw(float amount, string accountID)

{

cout << "Withdrawing " << amount << " from " << accountID << endl;

this->balance = this->balance - amount;

cout << "Your new balance is " << this->balance << endl;

}

void HighVolumeCheckingAccount::Deposit(float amount, string accountID)

{

cout << "Depositing " << amount << " to " << accountID << endl;

cout << "Transaction charge = " << DepositFee << endl;

cout << "Total Amount =" << amount - DepositFee << endl;

this->balance = this->balance + amount -DepositFee;

cout << "Your new balance is " << this->balance << endl;

}

#pragma once

//Money Market Account

//• Pays higher interest rate than any of the above accounts

//• Requires minimum balance for the account to start earning interests.

//• Withdrawals are limited(e.g.six per month).

#include "Account.h"

class MoneyMarketAccount: public Account

{

private:

float minimumBalance;

int withdrawalLimit = 6;

//float balance;

//std::set<std::string> documentContents;

//std::set<std::string> search;

//int searchLength;

//int documentLength;

//double simScore;

public:

MoneyMarketAccount();

void display();

//void AddInterest(float amount);

//void printWords(std::string newString);

//void addWords(std::string newString, std::set<std::string>& whatToAddTo);

//double ComputeSim();

//auto GetDocument() { return documentContents; }

//auto GetSimScore() { return simScore; }

};

#include "MoneyMarketAccount.h"

MoneyMarketAccount::MoneyMarketAccount()

{

this->accountType = "Money Market Account";

}

void MoneyMarketAccount::display()

{

cout << "This is Money MarketAccount" << endl;

}

#pragma once

#include "Account.h"

//1. Personal Account : Account owned by an individual for personal use.There are further

//different types of personal account.

//a.Savings Account

//• Allows holder to make deposits& withdrawals

//• Unlimited number of withdrawals& deposits can be made

//• Earn interest on the money.

//• Can transfer money to other accounts

class SavingsAccount : public Account

{

private:

int savings\_acc\_count = 1;

public:

SavingsAccount();

SavingsAccount(float balance, float interestRate, std::string SSN);

void display();

};

#include "SavingsAccount.h"

SavingsAccount::SavingsAccount(float balance, float interestRate, std::string SSN) : Account(balance, interestRate, SSN)

{

savings\_acc\_count++;

string s\_a\_c = to\_string(savings\_acc\_count);

this->accountID = accountID + 'P' + s\_a\_c + 'S';

interestRate = interestRate;

this->accountType = "Personal Savings Account";

}

SavingsAccount::SavingsAccount()

{

static int savingAccountNumber;

savingAccountNumber++;

string savingAccountString = to\_string(savingAccountNumber);

//string a\_c = to\_string(account\_count);

this->accountID = this->accountID + 'S' + savingAccountString;

//interestRate = interestRate;

}

void SavingsAccount::display()

{

cout << "This is Personal Savings Account"<<endl;

cout << "Account number: " << accountID << endl;

std::cout << "Balance: " << balance << endl;

cout << "Interest Rate: " << interestRate << endl;

cout << "SSN: " << SSN << endl << endl;

}

#pragma once

#include "Account.h"

#include <time.h>

#include <set>

#include <list>

//A customer can hold one or more accounts.These could be both Personal accounts and /or Business

//accounts.Furthermore, the customer can invoke any of the transactions that were detailed in Part I.

//Among other attributes, a customer would have an arrival time(when the customer joins the ATM queue),

//service time(how long he / she would use the ATM machine), exit time(when the customer in done).This

//will enable you to compute the waiting time for the customer.

class Customers

{

protected:

int customerID;

std::list<Account\*> accountsTest;

Account\* accounts[2];

int arrivalTime;

int serviceTime;

int exitTime;

int waitingTime;

public:

Customers();

Customers(int arrivalTime, int serviceTime);

void addAccount(int account, int accountType);

void calculateWaitingTime();

virtual void Display();

int getServiceTime();

void report();

//void createAccounts(int howMany);

};

#include "Customers.h"

#include "SavingsAccount.h"

#include "CheckingAccount.h"

#include "BusinessChecking.h"

#include "CODAccount.h"

#include "ForeignCurrencyAccount.h"

#include "HighVolumeCheckingAccount.h"

#include "MoneyMarketAccount.h"

#include "BusinessSavings.h"

#include <queue>

#include <fstream>

Customers::Customers()

{

static int customerCount;

customerCount++;

this->customerID = customerCount;

}

Customers::Customers(int arrivalTime, int serviceTime = 0)

{

static int customerCount;

customerCount++;

this->customerID = customerCount;

this->arrivalTime = arrivalTime;

this->serviceTime = serviceTime;

}

void Customers::addAccount(int account, int accountType)

{

/\*this->accounts.push\_back(account);\*/

//Account\* accounts[2];

if (accountType == 1)

{

accounts[account] = new SavingsAccount();

//accounts[amount] = new SavingsAccount();

}

else if (accountType == 2)

{

accounts[account] = new CheckingAccount();

//accounts[amount] = new CheckingAccount();

}

else if (accountType == 3)

{

accounts[account] = new MoneyMarketAccount();

//accounts[amount] = new MoneyMarketAccount();

}

else if (accountType == 4)

{

accounts[account] = new CODAccount();

//accounts[amount] = new CODAccount();

}

else if (accountType == 5)

{

accounts[account] = new BusinessSavings();

//accounts[amount] = new BusinessSavings();

}

else if (accountType == 6)

{

accounts[account] = new BusinessChecking();

//accounts[amount] = new BusinessChecking();

}

else if (accountType == 7)

{

accounts[account] = new HighVolumeCheckingAccount();

//accounts[amount] = new HighVolumeCheckingAccount();

}

else//foreignCurrency

{

accounts[account] = new ForeignCurrencyAccount();

//accounts[amount] = new ForeignCurrencyAccount();

}

this->accountsTest.push\_back(accounts[account]);

}

void Customers::calculateWaitingTime()

{

}

void Customers::Display()

{

//Account\* accounts[2];

cout << "Your customer ID is: " << customerID << endl;

cout << "Your service time is: " << this->serviceTime << endl;

cout << "Your accounts are: " << endl;

//accounts[i].

//for (int i = 0; i < 2; i++)

//{

// //cout << i << endl;

// try

// {

// if (accounts[i] == nullptr)

// {

// cout << "Testing";

//

// }

// else

// {

// accounts[i]->display();

// }

//

// }

// catch(const std::exception& e)

// {

// cout << e.what() << endl;

// }

// catch (char\* e)

// {

// cout << "Caught you" << endl;

// }

//}

//cout << "Your first account is " <<

//typedef std::unique\_ptr<Account> AccountPointer;

typedef std::list<Account\*>::iterator AccountPointer;

for (AccountPointer i = accountsTest.begin(); i != accountsTest.end(); i++)

{

(\*i)->display();

}

//for\_each(accountsTest.begin(), accountsTest.end(), &Account::display);

//for\_each()

}

int Customers::getServiceTime()

{

return serviceTime;

}

void Customers::report()

{

fstream myfile;

myfile.open("output.txt", fstream::app);

myfile << "Your customer ID is: " << customerID << endl;

myfile << "Your service time is: " << this->serviceTime << endl;

myfile << "Your accounts are: " << endl;

myfile.close();

cout << "Your customer ID is: " << customerID << endl;

cout << "Your service time is: " << this->serviceTime << endl;

cout << "Your accounts are: " << endl;

typedef std::list<Account\*>::iterator AccountPointer;

for (AccountPointer i = accountsTest.begin(); i != accountsTest.end(); i++)

{

//(\*i)->display();

(\*i)->reportDone();

}

}

#pragma once

#include <queue>

#include "Customers.h"

//contains a queue of customers

class ATM

{

private:

queue<Customers> customerQueue;

Customers lastCustomer;

int ATMID;

Customers currentCustomer;

protected:

public:

ATM();

void joinQueue(Customers customer);

void leaveQueue();

void setCurrentCustomer(Customers customer);

};

#include "ATM.h"

#include "Customers.h"

ATM::ATM()

{

static int ATMNumber;

ATMNumber++;

this->ATMID = ATMNumber;

}

//ATM::ATM()

//{

// Customers customer(1, 2);

// this->lastCustomer = customer;

//}

void ATM::joinQueue(Customers customer)

{

customerQueue.push(customer);

}

void ATM::leaveQueue()

{

this->lastCustomer = this->customerQueue.front();

this->customerQueue.pop();

}

void ATM::setCurrentCustomer(Customers customer)

{

this->currentCustomer = customer;

}

#pragma once

#include <string>

#include "ATM.h"

//e) Bank

//The bank is the entity that the above components belong to.It will read the user inputand generate

//appropriate objects e.g.the total number of ATM machines, the input file handle, etc.It can either read

//the input fileand pass the information to the appropriate components or , pass the filename to the

//components which process the relevant parts.Before control is passed to the System Controller, the

//Traffic Generator is called to create the customer baseand initial traffic.At the end of the simulation, the

//casino should invoke the appropriate methods of the Statistics Keeper to print the final report.

//mybank.set\_sim\_time(time);

//mybank.generate\_customerbase(); // Traffic Generator

//mybank.generate\_initial\_traffic(); // Traffic Generator

//mybank.simulate(); // System Controller

//mybank.report(); // Statistic Keeper

using namespace std;

class Bank

{

private:

int ATMS;

string inputFile;

int simulationTime;

ATM\* chosenATM;

list<ATM> atmList;

queue<Customers> customerQueue;

int customersServed;

protected:

public:

Bank(int ATMS, string inputFile);

void set\_inputfile(string inputFile);

void set\_atm\_num(int ATMs);

void set\_sim\_time(int time);

void generate\_customerbase(); // Traffic Generator

void report(); // Statistic Keeper

list<ATM> getATMList();

queue<Customers> getCustomers();

void addCustomer(Customers \*customer);

void serveCustomer();

int getCustomersServed();

};

#include "Bank.h"

//Documentation(out of 10) :

// Part 1 : Hierarchy Design & Implementation(out of 12) :

// Part 2 : System Components(out of 13) :

// Part 3 & 4 : Event Handling and Scheduling(out of 15) :

Bank::Bank(int ATMS, string inputFile)

{

this->ATMS = ATMS;

this->inputFile = inputFile;

this->customersServed = 0;

//this->trafficGenerator = TrafficGenerator();

for (int i = 0; i < ATMS; i++)

{

ATM atm = ATM();

this->atmList.push\_back(atm);

}

}

void Bank::set\_inputfile(string inputFile)

{

this->inputFile = inputFile;

}

void Bank::set\_atm\_num(int ATMs)

{

this->ATMS = ATMs;

}

void Bank::set\_sim\_time(int time)

{

this->simulationTime = time;

}

void Bank::generate\_customerbase()

{

//TrafficGenerator trafficGenerator = TrafficGenerator();

}

void Bank::report()

{

//cout << "Duration of simulation: " << this->trafficGenerator.getTime() << endl;

cout << "Number of ATMs : " << this->ATMS << endl;

}

list<ATM> Bank::getATMList()

{

return this->atmList;

}

queue<Customers> Bank::getCustomers()

{

return this->customerQueue;

}

void Bank::addCustomer(Customers\* customer)

{

this->customerQueue.push(\*customer);

}

void Bank::serveCustomer()

{

this->customersServed++;

Customers customer = customerQueue.front();

customer.report();

this->customerQueue.pop();

}

int Bank::getCustomersServed()

{

return this->customersServed;

}

#include "Partition.h"

//#include "TrafficGenerator.h"

#include "Customers.h"

#include "Bank.h"

#pragma once

class TimingWheel

{

private:

int time;

int maxDelay = 12;

//Partition \*slot[maxDelay[1]+1];

//TrafficGenerator traffic;

int currentSlot;

queue<Customers> customerQueue;

list<ATM> atmList;

vector<list<ATM>> timeSlots;

Bank\* bank;

int customersServed;

public:

TimingWheel(Bank\* bank);

void insert(Customers customer, ATM\* ATMPointer);

void schedule();

void createTimeSlots();

void addTime();

int getTime();

};

#include "TimingWheel.h"

TimingWheel::TimingWheel(Bank\* bank)

{

this->bank = bank;

this->time = 0;

this->createTimeSlots();

}

void TimingWheel::insert(Customers customer, ATM\* ATMPointer)

{

ATMPointer->joinQueue(customer);

}

void TimingWheel::schedule()

{

//Next TimeSlot in the list

int size = timeSlots.size();

this->atmList = timeSlots.back();

//This is customers using the atm

while (atmList.empty() == false)

{

//Take the next customer in the queue

customerQueue = bank->getCustomers();

Customers nextCustomer = customerQueue.front();

bank->serveCustomer();

//give the ATM the customer

list<ATM> currentTimeslotATMList = timeSlots.back();

ATM currentATM = currentTimeslotATMList.front();

currentATM.setCurrentCustomer(nextCustomer);

//add the atm to its new TimeSlot

int waitingTime = nextCustomer.getServiceTime();

//cout << "Waiting time" << waitingTime << endl;

list<ATM> nextATMList = timeSlots[waitingTime];

//cout << "\*" << timeSlots[waitingTime].size() << endl;

//cout << "\*" << nextATMList.size() << endl;

nextATMList.push\_back(currentATM);

timeSlots[waitingTime] = nextATMList;

atmList.pop\_front();

//cout << "\*\*" << timeSlots[waitingTime].size() << endl;

//cout << "\*\*" << nextATMList.size() << endl;

}

//cout << "\*\*\*" << atmList.size() << endl;

//move the timeslots forward

list<ATM> emptyATMList;

//cout << "Timing Wheel before pop and insert" << endl;

////for (int i = 0; i < timeSlots.size(); i++)

////{

//// cout << timeSlots[i].size() << endl;

////}

//cout << endl;

timeSlots.pop\_back();

timeSlots.insert(timeSlots.begin(), emptyATMList);

//for (int i = 0; i < timeSlots.size(); i++)

//{

// cout << timeSlots[i].size() << endl;

//}

}

void TimingWheel::createTimeSlots()

{

//get atms from the bank

this->atmList = bank->getATMList();

timeSlots.push\_back(atmList);

for (int i = 0; i < maxDelay -1; i++)

{

list<ATM> emptyATMList;

timeSlots.push\_back(emptyATMList);

}

}

void TimingWheel::addTime()

{

this->time++;

}

int TimingWheel::getTime()

{

return this->time;

}

#pragma once

#include "Bank.h"

//c) Statistics Keeper

//This component collects all the data needed for reporting statistics.The simulation must run for the time

//specified by the user(we’ll use one iteration of the loop as one unit of time) and report statistics at the

//end of the simulation.Some information provided might include :

//1. Duration of simulation

//2. Number of ATM machines

//3. Total no.of customers serviced

//4. Total no.of customers serviced categorized by type of accounts

//5. Average service time for each customer

//6. Average waiting time for each customer

//7. Total number of transactions

//8. Total number of transactions categorized by type of transaction

//9. Total amount of money deposited / withdrawn

//10. Number of refill\_cash events generated

class StatisticsKeeper

{

private:

int durationOfSimulation;

int numberOfATMMachines;

int numberOfCustomersServed;

int numberOfCustomersServicedByAccountType[8] = { 0,0,0,0,0,0,0,0 };

float averageServiceTimePerCustomer;

float waitingServiceTimePerCustomer;

int totalNumberOfTransactions;

int totalNumberOfTransactionsByTransactionType[4] = { 0,0,0,0 };

int totalAmountOfMoneyDeposited;

int totalAmountOfMoneyWithdrawn;

int numberOfRefillCashEventsGenerated;

int totalServiceTime;

int numberOfPeopleServiced;

Bank\* bank;

protected:

public:

StatisticsKeeper(Bank\* bank);

void Report();

void setDurationOfSimulation(int time);

void getNumberOfATMs();

void addCustomerServed();

void sumServiceTime(int Time);

void printFinalStatus();

};

#include "StatisticsKeeper.h"

#include <iostream>

#include <fstream>

StatisticsKeeper::StatisticsKeeper(Bank\* bank)

{

this->bank = bank;

this->getNumberOfATMs();

this->numberOfCustomersServed = 0;

this->totalServiceTime = 0;

this->numberOfPeopleServiced = 0;

}

void StatisticsKeeper::Report()

{

cout << "Number of ATMS: " << this->numberOfATMMachines << endl;

cout << "Duration of Simulation: " << this->durationOfSimulation << endl;

cout << "Customers served: " << this->bank->getCustomersServed() << endl;

//cout << "Customers served by account type: " << endl;

cout << "Average Service Time: " << this->averageServiceTimePerCustomer << endl;

//cout << "Average Waiting Time: " << endl;

cout << endl;

fstream myfile;

myfile.open("output.txt", fstream::app);

myfile << "Number of ATMS: " << this->numberOfATMMachines << endl;

myfile << "Duration of Simulation: " << this->durationOfSimulation << endl;

myfile << "Customers served: " << this->bank->getCustomersServed() << endl;

//cout << "Customers served by account type: " << endl;

myfile << "Average Service Time: " << this->averageServiceTimePerCustomer << endl;

//cout << "Average Waiting Time: " << endl;

myfile << endl;

myfile.close();

}

void StatisticsKeeper::setDurationOfSimulation(int time)

{

this->durationOfSimulation = time;

}

void StatisticsKeeper::getNumberOfATMs()

{

this->numberOfATMMachines = bank->getATMList().size();

}

void StatisticsKeeper::addCustomerServed()

{

this->numberOfCustomersServed = this->bank->getCustomersServed();

}

void StatisticsKeeper::sumServiceTime(int Time)

{

this->totalServiceTime += Time;

this->numberOfPeopleServiced++;

this->averageServiceTimePerCustomer = this->totalServiceTime / this->numberOfPeopleServiced;

}

void StatisticsKeeper::printFinalStatus()

{

cout << endl;

cout << "Final Status: " << endl;

cout << "Number of ATMS: " << this->numberOfATMMachines << endl;

cout << "Duration of Simulation: " << this->durationOfSimulation << endl;

cout << "Customers served: " << this->bank->getCustomersServed() << endl;

//cout << "Customers served by account type: " << endl;

cout << "Average Service Time: " << this->averageServiceTimePerCustomer << endl;

//cout << "Average Waiting Time: " << endl;

cout << endl;

fstream myfile;

myfile.open("output.txt", fstream::app);

myfile << "Number of ATMS: " << this->numberOfATMMachines << endl;

myfile << "Duration of Simulation: " << this->durationOfSimulation << endl;

myfile << "Customers served: " << this->bank->getCustomersServed() << endl;

//cout << "Customers served by account type: " << endl;

myfile << "Average Service Time: " << this->averageServiceTimePerCustomer << endl;

//cout << "Average Waiting Time: " << endl;

myfile << endl;

myfile.close();

}

#pragma once

#include <set>

#include "Customers.h"

#include <list>

#include "Bank.h"

#include "TimingWheel.h"

#include "StatisticsKeeper.h"

//This module generates the customer base, initial trafficand dynamic player traffic.The customer base is

//the total number of customers that hold accounts in the bank.The initial traffic is the number of customers

//in line to use the ATM when simulation starts.Dynamic player traffic are the customers that will arrive

//during the simulation.On arrival, the customer joins the shortest queue(once we scale up in the next

// phases).

// You can also choose to have NO initial traffic assuming you are starting the simulation when the ATM is

// not being used.

// The above is generated based on the user specified parameters which will come in form of an input file of

// the given format(black – actual content of file; blue – explanation) :

// In the above file, it is specified that there is a total of 200 customers.Out of the them, five should be

// randomly chosen to be in the queue as initial traffic(if implemented).After this we generate a random

// number between 0 and 2 to decide how many customers arrive at a given time.For every customer, you

// then generate a random number between 5 and 12 to decide on the service time.Next you generate

// another random number between 0 and 100. If it is less than 20, the customer has one account; multiple,

// otherwise.You are free to implement how many accounts(if multiple) in any way you want.At this point,

// 200 <= customer base

// 5 <= initial traffic(if used); change the number to 20 for Phase IV

// 2 <= 0 – 2 customers can come at any given instant – generated randomly; change the number to 7 for Phase IV

// 5 12 <= a customer’s service time can vary from 5 – 12 time units

// 80 <= 80 % of the time a customer has multiple accounts – you are free to decide how many

// Personal 65 <= 65 % of accounts are personal

// Business 35 <= 35 % of accounts are business

// Savings 50 30 <= 50 % of personal accounts are Savings account; 30 % of business accounts are Savings account

// Checking 30 55

// Money Market 8 0

// Certificate of Deposits 12 0

// High Volume Checking 0 10

// Foreign Currency 0 5

// CS 501 Fall 2020

// Project Part II University of Bridgeport

// what is left is to decide what is the type of each account.For this, you’ll generate two more random

// numbers between 0 and 100, one to decide whether the account is Personal or Business and the second

// to decide on the subtype.Now you are ready to create the appropriate account(s) for the customer.Those

// of us that are implementing a scaled down version can modify the numbers for the subtypes.Please do

// NOT change the format of the file.Finally, you’ll decide how manyand what kind of transactions will the

// customer perform on the account – you are free to design your own methodology for that.

// At simulation time, the user will be prompted to specify the name of the input fileand the number of ATM

// machines(you can specify 1 for now) and the duration of simulation.

class TrafficGenerator

{

private:

int time = 0;

int customerBase = 200;

int initialTraffic = 5;

int customersAtATime = 2;

int serviceTime[2] = { 5,12 };

float multipleAccounts = 80;

float personalAccounts = 65;

float businessAccounts = 100 - personalAccounts;

int savingAccounts[2] = { 50, 30 };

int checkingAccounts[2] = { 30,55 };

int moneyMarketAccounts[2] = { 8,0 };

int certificateOfDeposits[2] = { 12,0 };

int highVolumeChecking[2] = { 0,10 };

int foreignCurrency[2] = { 0,5 };

std::list<Customers> customers;

Customers\* customerList[200];

int customersLeft;

Bank\* bank;

Customers\* currentCustomer;

TimingWheel\* timingWheel;

StatisticsKeeper\* statisticsKeeper;

protected:

public:

TrafficGenerator(Bank\* bank, TimingWheel\* timingWheel);

void CreateCustomer(int amount);

void CreateAccounts(int amount);

int getTime();

int\* getServiceTime();

void RunSimulation();

};

#include "TrafficGenerator.h"

#include <cstdlib>

#include "SavingsAccount.h"

#include "CheckingAccount.h"

#include "BusinessChecking.h"

#include "CODAccount.h"

#include "ForeignCurrencyAccount.h"

#include "HighVolumeCheckingAccount.h"

#include "MoneyMarketAccount.h"

#include "BusinessSavings.h"

#include "TimingWheel.h"

TrafficGenerator::TrafficGenerator(Bank\* bank, TimingWheel\* timingWheel)

{

this->timingWheel = timingWheel;

this->bank = bank;

this->customersLeft = customerBase;

this->statisticsKeeper = new StatisticsKeeper(bank);

}

void TrafficGenerator::CreateCustomer(int amount)

{

//Customers\* customerList[200];

int accountAmount = 0;

for (amount; amount > 0; amount--)

{

//int serviceTime[2] = { 5,12 };

int randomServiceTime = rand() % 6 + 5;

this->statisticsKeeper->sumServiceTime(randomServiceTime);

//cout << "Random service time" << randomServiceTime << endl;

this->customersLeft--;

//customerList[this->customersLeft] = new Customers(this->time, randomServiceTime);

this->currentCustomer = new Customers(this->time, randomServiceTime);

//Customers\* customer = new Customers(time);

//Customers customer(this->time, this->time);

int randomNumber = (rand() % 100 + 1);

if (randomNumber <= this->multipleAccounts)

{

accountAmount = 2;

}

else

{

accountAmount = 1;

}

CreateAccounts(accountAmount);

bank->addCustomer(this->currentCustomer);

//customer.addAccount();

//this->customers.push\_back(customer);

}

}

void TrafficGenerator::CreateAccounts(int amount)

{

int randomNumberAccount = (rand() % 100 + 1);

//Account\* accounts[2];

//Account account;

for (amount; amount > 0; amount--)

{

//Customers\* customer = new Customers(time);

//Customers customer(time, this->time);

if (randomNumberAccount <= this->personalAccounts)

{

int randomNumber = (rand() % 100 + 1);

int personalSavingAccountPercent = this->savingAccounts[0];

int personalCheckingAccountPercent = personalSavingAccountPercent + this->checkingAccounts[0];

int personalMoneyMarketAccountPercent = personalCheckingAccountPercent + this->moneyMarketAccounts[0];

if (randomNumber <= personalSavingAccountPercent)

{

this->currentCustomer->addAccount(amount -1, 1);

//customerList[this->customersLeft]->addAccount(amount-1, 1);

//account = SavingsAccount();

//accounts[amount] = new SavingsAccount();

}

else if(randomNumber <= personalSavingAccountPercent)

{

this->currentCustomer->addAccount(amount - 1, 2);

//customerList[this->customersLeft]->addAccount(amount-1, 2);

//account = CheckingAccount();

//accounts[amount] = new CheckingAccount();

}

else if (randomNumber <= personalMoneyMarketAccountPercent)

{

this->currentCustomer->addAccount(amount - 1, 3);

//customerList[this->customersLeft]->addAccount(amount-1, 3);

//account = MoneyMarketAccount();

//accounts[amount] = new MoneyMarketAccount();

}

else //COD

{

this->currentCustomer->addAccount(amount - 1, 4);

//customerList[this->customersLeft]->addAccount(amount-1, 4);

//account = CODAccount();

//accounts[amount] = new CODAccount();

}

}

else //businessAccount

{

int randomNumber = (rand() % 100 + 1);

int businessSavingAccountPercent = this->savingAccounts[1];

int businessCheckingAccountPercent = businessSavingAccountPercent + this->checkingAccounts[1];

int businessHighVolumeCheckingAccountPercent = businessCheckingAccountPercent + this->highVolumeChecking[1];

if (randomNumber <= businessSavingAccountPercent)

{

this->currentCustomer->addAccount(amount - 1, 5);

//customerList[this->customersLeft]->addAccount(amount-1, 5);

//account = BusinessSavings();

//accounts[amount] = new BusinessSavings();

}

else if (randomNumber <= businessCheckingAccountPercent)

{

this->currentCustomer->addAccount(amount - 1, 6);

//customerList[this->customersLeft]->addAccount(amount-1, 6);

//account = BusinessChecking();

//accounts[amount] = new BusinessChecking();

}

else if (randomNumber <= businessHighVolumeCheckingAccountPercent)

{

this->currentCustomer->addAccount(amount - 1, 7);

//customerList[this->customersLeft]->addAccount(amount-1, 7);

//account = HighVolumeCheckingAccount();

//accounts[amount] = new HighVolumeCheckingAccount();

}

else //foreignCurrency

{

this->currentCustomer->addAccount(amount - 1, 8);

//customerList[this->customersLeft]->addAccount(amount-1, 8);

//account = ForeignCurrencyAccount();

//accounts[amount] = new ForeignCurrencyAccount();

}

//customer.addAccount(accounts[amount]);

}

//this->currentCustomer->Display();

//customerList[this->customersLeft]->addAccount(account);

//customerList[this->customersLeft].addAccount(account);

//this->customers.push\_back(customer);

}

}

int TrafficGenerator::getTime()

{

return this->time;

}

int\* TrafficGenerator::getServiceTime()

{

return this->serviceTime;

}

void TrafficGenerator::RunSimulation()

{

CreateCustomer(initialTraffic);

int customer = 0;

while (customersLeft > 0)

{

this->timingWheel->addTime();

this->time = this->timingWheel->getTime();

this->statisticsKeeper->setDurationOfSimulation(this->time);

this->timingWheel->schedule();

this->statisticsKeeper->Report();

int howManyCustomers = rand() % 3;

//cout << "Creating " << howManyCustomers << " Customer[s] " << "We have " << customersLeft << "Customers left" << endl;

this->customersLeft - howManyCustomers;

CreateCustomer(howManyCustomers);

}

//typedef std::list<Customers>::iterator CustomerPointer;

//for (CustomerPointer i = customers.begin(); i != customers.end(); i++)

//{

// i->Display();

//}

//Customers customer;

//this->bank->getCustomers().back().Display();

//this->bank->getCustomers().front().Display();

this->statisticsKeeper->printFinalStatus();

//this->customerList[1]->Display();

//this->customerList[10]->Display();

//this->customerList[100]->Display();

//cout << customers.size() << endl;

}

#pragma once

#include <iostream>

#include "Account.h"

#include "SavingsAccount.h"

#include "CheckingAccount.h"

#include "BusinessChecking.h"

#include "BusinessSavings.h"

#include "CODAccount.h"

#include "ForeignCurrencyAccount.h"

#include "HighVolumeCheckingAccount.h"

#include "MoneyMarketAccount.h"

#include "PersonalAccount.h"

void accountTest(Account& account);

void testingRegularAccount();

void testingSavingsAccount();

void testingCheckingAccount();

void testAccounts();

#include "driver.h"

#include "TrafficGenerator.h"

#include "Bank.h"

using namespace std;

int main()

{

//testingRegularAccount();

//testingSavingsAccount();

//testingCheckingAccount();

//---------------------------------------------------------------------------------------------------------------------------------

//static int test;

//test = test + 1;

//cout << test << endl;

//testAccounts();

//--------------------------------------------------------------------------------------------------------------------------------

//TrafficGenerator trafficGenerator = TrafficGenerator();

int ATMs, time;

ATMs = 2;

string fname = "test string";

Bank\* mybank = new Bank(ATMs, fname);

//cout << "Please specify input file name:\n";

//cin >> fname;

//mybank.set\_inputfile(fname);

//cout << "Number of ATMs:\n";

//cin >> ATMs; // input 1 for this part

//mybank.set\_atm\_num(ATMs);

//cout << "Simulation time:\n";

//cin >> time;

time = 200;

//mybank.set\_sim\_time(time);

//mybank.generate\_customerbase(); // Traffic Generator

//mybank.generate\_initial\_traffic(); // Traffic Generator

//mybank.simulate(); // System Controller

//mybank.report(); // Statistic Keeper

TimingWheel\* timingWheel = new TimingWheel(mybank);

TrafficGenerator trafficGenerator = TrafficGenerator(mybank, timingWheel);

trafficGenerator.RunSimulation();

return 0;

}

void testAccounts()

{

Account\* accounts[8];

accounts[0] = new SavingsAccount();

accounts[1] = new CheckingAccount();

accounts[2] = new MoneyMarketAccount();

accounts[3] = new CODAccount();

accounts[4] = new BusinessSavings();

accounts[5] = new BusinessChecking();

accounts[6] = new HighVolumeCheckingAccount();

accounts[7] = new ForeignCurrencyAccount();

Account acc;

acc.display();

for (int i = 0; i < 7; i++)

{

accounts[i]->display();

}

}

void testingCheckingAccount()

{

cout << "Testing Checking Account" << endl;

CheckingAccount checkingAccount(0, "000");

accountTest(checkingAccount);

}

void testingSavingsAccount()

{

cout << "Testing Savings Account" << endl;

SavingsAccount savingsAccount(0, 1.1, "000");

accountTest(savingsAccount);

}

void testingRegularAccount()

{

cout << "Testing Account" << endl;

Account account(0, 1.1, "000");

accountTest(account);

}

void accountTest(Account& account)

{

cout << account;

account.Deposit(100, "1234");

account.AddInterest();

account.Withdraw(50, "1234");

cout << account;

}