**C++ Workshop – 150018**

**Homework Assignment #4**

**Friends, Static Methods, Exceptions**

**General:**

* Pay attention to readability and indentation, of program.
* Do exactly what you are asked for in each question.
* Define function according to need.
* Don't forget meaningful names for variables, document code and functions, and add running examples for your program at end.
* Homework is given in individually.
* You have one week to give in your homework.

In this assignment you will write a program for an ATM machine. You will do it in stages until the complete program is written.

In all cases when an error occurs you should throw an appropriate exception and continue to the next transaction. The list or error messages is explained later in each part of the program.

1. Define the class **Clock** which contains the following fields:
   * **hour** (0-23)
   * **minute** (0-59)
   * **second** (0-59)

(all fields should be initialized at time of definition with 0)

Add the following methods to the class:

* constructors
  + default constructor that sets the time to 00:00:00
  + constructor with parameters that receives values and assigns them to its fields. In the case of illegal values, the time is set to 00:00:00
  + copy constructor
* **set/get** for each field in the class
* operator **+=** that receives the number of seconds and updates the time accordingly
* operator **<<** that prints the time in the format hh:mm:ss. Note, even when a value is less than 10, it is still printed as a two digit number.
* operator **>>** that inputs a time in the above format. In the case of illegal input, the program throws an exception and the object is assigned 00:00:00

In cases when an error occurs, the program should throw an exception with the appropriate message and enter the time 00:00:00 in the object.:

* Wrong time format.
* Invalid time – more than 60 seconds
* Invalid time – more than 60 minutes.
* Invalid time – more than 24 hours.
* Invalid time – negative number of seconds.
* Invalid time – negative number of minutes.
* Invalid time – negative number of hours.

1. Define the class **Account** to represent bank accounts. It contains the fields:
   * **accountNumber**
   * **code** (password of 4 digits, the leftmost digit being non-0)
   * **balance**
   * **mail** (mail of account owner – of type string – a correct email address consists of @ with preceding characters with no space, and after the @ sign, characters with no space, ending with one of the following: .com or .co.il. you MUST use the string class taught in class, defined in the STL library  
     <https://www.cplusplus.com/reference/string/string> ).

Add the following methods to the class:

* constructors
  + default constructor that initializes all fields to 0,and empty string in the email field.
  + constructor with parameters that receives values and assigns them to its fields
* **set/get** (account number and password code cannot be reset)
* operator **>>** for inputting initial account data for account number password and mail. The initial customer balance is 0.
* method **withdraw(int nis)** which withdraws **nis** from the account. Cash can be withdrawn up to a sum of 2500 NIS at one time, with a credit limit of 6000 NIS (overdraft of up to 6000 NIS will be allowed).
* method **deposit(int nis)** which deposits checks of up to 10000 NIS into the account.

In addition, the class contains the following static methods and fields:

* **sumWithdraw** – sum of all bank withdrawals (from all accounts)
* **sumDeposit** – sum of all check deposits (into all accounts)
* **getSumWithdraw()** – method that returns the sum of all bank withdrawals
* **getSumDeposit()** – method that returns the sum of all check deposits

In cases when an error occurs, you should throw an exception with an appropriate message:

* ERROR: wrong code!
* ERROR: wrong email!
* ERROR: cannot deposit more than 10000 NIS!
* ERROR: cannot withdraw more than 2500 NIS!
* ERROR: cannot have less than - 6000 NIS!

1. You are given the main program below for the ATM. The program uses the classes defined in the previous questions. The program assumes that the bank has 10 accounts.

In the first stage the user is asked to initialize the data for all the accounts:

* account number: must be unique (i.e., no two accounts can have the same number)
* password: a number of 4 digits, where the first digit is not 0
* the user does not initialize the balance which is set to 0
* email address as defined above.

The exceptions which can occur at this stage are:

* ERROR: code must be of 4 digits!
* ERROR: account number must be unique!
* ERROR: email must contain @!
* ERROR: email must end at .com or .co.il!

Afterwards, the program enters a loop and processes a transaction until it receives a quit (0) request. The program performs the transaction and prints an appropriate message including the time the transaction started. In the case when an exception occurs, the program prints the time (before the transaction is performed) and an error message and continues to process the next transaction.

Assume that the ATM opens at 08:00:00.

Checking a balance takes 20 seconds.

Withdrawals take 50 seconds

Deposits take 30 seconds

Getting the total sum of withdrawals or deposits each take 1 minute

Assume that all transactions are performed in succession without a break

The program performs correctly as long as no exception occurred.

You should add handling all possible exceptions to the program. The throwing is done in the methods only, while the catching is done in the main program. Which means you must add the catching options to the following main program as well.

#include <iostream>

#include "Clock.h"

#include "Account.h"

using namepace std;

enum action

{

stop,

balance,

deposit,

withdraw,

sumDeposit,

sumWithdraw

};

action menu()

{

cout << "enter 1 to get account balance" << endl;

cout << "enter 2 to deposit money" << endl;

cout << "enter 3 to withdraw money" << endl;

cout << "enter 4 to see the sum of all withdraws" << endl;

cout << "enter 5 to see the sum of all Deposits" << endl;

cout << "enter 0 to stop" << endl;

int x;

cin >> x;

return (action)x;

}

int findAccount(Account\* bank, int size)

{

int number, code;

cout << "please enter account number: ";

cin >> number;

int i = 0;

while (i < size && bank[i].getAccountNumber() != number)

i++;

cout << "please enter the code: ";

cin >> code;

if (bank[i].getCode() == code)

return i;

}

void printTransaction(Account a, action ac, Clock& c)

{

cout << c << "\t";

;

switch (ac)

{

case balance: cout << "account #: " << a.getAccountNumber() << "\t";

cout << "balance: " << a.getBalance() << endl;

break;

case deposit:

case withdraw: cout << "account #: " << a.getAccountNumber() << "\t";

cout << "new balance: " << a.getBalance() << endl;

break;

case sumDeposit:

cout << "sum of all deposits: " << Account::getSumDeposit() << endl;

break;

case sumWithdraw:

cout << "sum of all withdraws: " << Account::getSumWithdraw() << endl;

break;

}

}

void getBalance(Account\* bank, int size, Clock& c)

{

int i = findAccount(bank, size);

c += 20;

printTransaction(bank[i], balance, c);

}

void cashDeposit(Account\* bank, int size, Clock& c)

{

int i = findAccount(bank, size);

float amount;

cout << "enter the amount of the check: ";

cin >> amount;

bank[i].deposit(amount);

c += 30;

printTransaction(bank[i], deposit, c);

}

void cashWithdraw(Account\* bank, int size, Clock& c)

{

int i = findAccount(bank, size);

float amount;

cout << "enter the amount of money to withdraw: ";

cin >> amount;

bank[i].withdraw(amount);

c += 50;

printTransaction(bank[i], withdraw, c);

}

int main()

{

Clock c(8);

Account bank[10];

cout << "enter account number and code for 10 accounts:\n";

for (int i = 0; i < 10; i++)

{

try {

cin >> bank[i];

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

if (bank[i].getAccountNumber() == bank[j].getAccountNumber())

throw "ERROR: account number must be unique!\n";

}

catch (const char\* msg)

{

cout << c << '\t' << msg;

i--;

}

}

action ac = menu();

while (ac)

{

switch (ac)

{

case balance: getBalance(bank, 10, c);

break;

case withdraw:cashWithdraw(bank, 10, c);

break;

case deposit:cashDeposit(bank, 10, c);

break;

case sumDeposit:c += 60;

printTransaction(bank[0], sumDeposit, c);

break;

case sumWithdraw:c += 60;

printTransaction(bank[0], sumWithdraw, c);

}

ac = menu();

}

return 0;

}