**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 may be done in teams of two.
* 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.

Suggestion: It is best to first write the code assuming that everything is in order and then afterwards add the code that throws exceptions when there is an error.

As a general rule, exceptions are thrown by functions and are caught in the main program.

Keep in mind then in addition to writing the classes that are described in this assignment, you are also to make changes to the main program (aka add *try*s and *catch*s)

Reminder: If you throw a string of type rvalue (aka a value that can be assigned to a variable but we cannot assign it a value – for example the cstring “Wrong time format”) then you must catch it in a variable of type const char \*

**Part A.**

Define the class **Clock**

**a.** Declare the following fields:

* + **hour** (0-23)
  + **minute** (0-59)
  + **second** (0-59)

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

b. Add the following constructors to the class:

* constructors with parameters that receives values and assigns them to its fields.

In the case that even one of the values is illegal,

* + The time is set to 00:00:00 (aka do not change the default values of the fields)
  + Throw an appropriate exception that will be caught in the main program. Note that there can be many causes for the error. Your program will only print out at most one error. It should check for errors and throw them in the following order:

Invalid time - negative number of seconds.

Invalid time - more than 60 seconds.

Invalid time - negative number of minutes.

Invalid time - more than 60 minutes.

Invalid time - negative number of hours.

Invalid time - more than 24 hours.

* + In other words if will first check if there is an error in the seconds. If there is an error with the seconds then it will print the appropriate message.
  + If there is no error in the seconds, then it will check if there is an error in the minutes. If there is an error in the minutes, ten it will print an appropriate message.
  + If there is not an error in the seconds nor in the minutes then it will check if there was an error in the hours. If there is an error in the hours then it will print an appropriate message.
* ***Food for thought : why doesn’t it need a copy*** ***constructor***?

c. Add the following methods to the class:

* **set/get** for each field in the class
  + If there is an erroneous value is passed to the setters, then the attribute’s value will not be changed and an error message will be thrown and must be printed in the main program.
* **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 according to the following format:

Invalid time - negative number of seconds.

Invalid time - more than 60 seconds.

Invalid time - negative number of minutes.

Invalid time - more than 60 minutes.

Invalid time - negative number of hours.

Invalid time - more than 24 hours.

If an error occurs that doesn’t fall in any of the above categories then the exception thrown should be:

Wrong time format.

**Part B.**

a. 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**
  + **email** (email 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> ).

b. Add the following constructors to the class:

* **empty 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

If there is even one erroneous value, then,

* **All of the fields** will remain with their default values
* An appropriate error will be thrown. The main program will catch the error and print it.
* If there is more than one error, only one error message will be thrown.
* Errors will be checked according to the following order:
  + If there is a problem with the code then it will throw the following error:

ERROR: code must be of 4 digits!

* + If the code is fine but there is a problem with the email where the email doesn’t contain a @ sign then it will throw the following error:

ERROR: email must contain @!

* + If the code is fine and there is no problem with the @ sign but the extension is not .com nor .co.il then it will throw the following error:

ERROR: email must end at .com or .co.il!

c. Add the following methods to the class:

* **set/get** (account number and password code cannot be reset. Values for these fields can only be assigned through the constructor or via input. )
* operator **>>** for inputting initial account data for account number, password, and email. The initial customer balance is 0.

If there is even one erroneous value, then,

* **All of the fields** will remain with their previous values.
* An appropriate error will be thrown. The main program will catch the error and print it.
* If there is more than one error, only one error message will be thrown.
* Errors will be checked according to the following order:
  + If there is a problem with the code then it will throw the following error:

ERROR: code must be of 4 digits!

* + If the code is fine but there is a problem with the email where the email doesn’t contain a @ sign then it will throw the following error:

ERROR: email must contain @!

* + If the code is fine and there is no problem with the @ sign but the extension is not .com nor .co.il then it will throw the following error:

ERROR: email must end at .com or .co.il!

* **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). You program should first check if the withdrawal would cause an overdraft of more than 6000 NIS and if so throw an appropriate exception. If the withdrawal will not cause a problem of overdraft then your program should check if the withdrawal amount is over 2500 NIS and if so then it should throw an appropriate exception. If everything is in order then it should continue with the transaction. (Obviously if there was an error then the transaction would have been cancelled.)
* **deposit(int)** which deposits checks of up to 10000 NIS into the account. If the deposit is for more than 10000 NIS then an exception should be thrown and the transaction should be cancelled.

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

* **sumWithdraw** – field: sum of all bank withdrawals (from all accounts)
* **sumDeposit** – field: 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!

**Part C.**

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 STOP (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

**Please Note:**

The main program as it is written (and the classes that you are to write) all run fine as long as there is no errors.

**Exceptions are thrown from methods** (they are not written in the main programming they are **caught** in the main program.) You are to throw exceptions at the **beginning of the method**. You are to catch the exceptions in the main program.

It is therefore up to you , to look over the main program that you are being given and decide what to put in try blocks and where to add catch blocks.

**(For the example we only included one try-catch block. You are to figure out where to include the other blocks.)**

#include <iostream>

#include "Clock.h"

#include "Account.h"

using namespace std;

enum ACTION {

STOP,

BALANCE,

DEPOSIT,

WITHDRAW,

SUM\_DEPOSIT,

SUM\_WITHDRAW

};

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 deposits" << endl;

cout << "enter 5 to see the sum of all withdrawals" << 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:\n";

cin >> number;

int i = 0;

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

i++;

if (i >= size)

throw "ERROR: no such account number\n";

cout << "please enter the code:\n";

cin >> code;

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

return i;

throw "ERROR: wrong code!\n";

}

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 SUM\_DEPOSIT:

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

break;

case SUM\_WITHDRAW:

cout << "sum of all withdrawals: " << 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 deposit:\n ";

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:\n ";

cin >> amount;

bank[i].withdraw(amount);

c += 50;

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

}

void checkAccount(Account bank[], int i) {

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

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

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

}

int main() {

const int SIZE = 10;

Clock c(8, 0, 0);

Account bank[SIZE];

cout << "enter account number, code and email for " << SIZE << " accounts:\n";

for (int i = 0; i < SIZE; i++) {

try {

cin >> bank[i];

checkAccount(bank, i);

}

catch (const char\* msg) {

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

i--;

}

}

ACTION ac = menu();

while (ac) {

switch (ac) {

case BALANCE: getBalance(bank, SIZE, c);

break;

case WITHDRAW:cashWithdraw(bank, SIZE, c);

break;

case DEPOSIT:cashDeposit(bank, SIZE, c);

break;

case SUM\_DEPOSIT:c += 60;

printTransaction(bank[0], SUM\_DEPOSIT, c);

break;

case SUM\_WITHDRAW:c += 60;

printTransaction(bank[0], SUM\_WITHDRAW, c);

}

ac = menu();

}

return 0;

}

The following is an example of a program that runs that had no errors. The bank is this program only has 3 accounts.

enter account number, code and email for 3 accounts:

123 4444 me@gmail.com

234 5555 you@walla.co.il

345 6666 us@g.com

enter 1 to get account balance

enter 2 to deposit money

enter 3 to withdraw money

enter 4 to see the sum of all deposits

enter 5 to see the sum of all withdrawals

enter 0 to stop

2

please enter account number:

234

please enter the code:

5555

enter the amount of the deposit:

5000

08:00:30 account #: 234 new balance: 5000

enter 1 to get account balance

enter 2 to deposit money

enter 3 to withdraw money

enter 4 to see the sum of all deposits

enter 5 to see the sum of all withdrawals

enter 0 to stop

3

please enter account number:

234

please enter the code:

5555

enter the amount of money to withdraw:

1000

08:01:20 account #: 234 new balance: 4000

enter 1 to get account balance

enter 2 to deposit money

enter 3 to withdraw money

enter 4 to see the sum of all deposits

enter 5 to see the sum of all withdrawals

enter 0 to stop

1

please enter account number:

234

please enter the code:

5555

08:01:40 account #: 234 balance: 4000

enter 1 to get account balance

enter 2 to deposit money

enter 3 to withdraw money

enter 4 to see the sum of all deposits

enter 5 to see the sum of all withdrawals

enter 0 to stop

2

please enter account number:

345

please enter the code:

6666

enter the amount of the deposit:

2000

08:02:10 account #: 345 new balance: 2000

enter 1 to get account balance

enter 2 to deposit money

enter 3 to withdraw money

enter 4 to see the sum of all deposits

enter 5 to see the sum of all withdrawals

enter 0 to stop

3

please enter account number:

345

please enter the code:

6666

enter the amount of money to withdraw:

500

08:03:00 account #: 345 new balance: 1500

enter 1 to get account balance

enter 2 to deposit money

enter 3 to withdraw money

enter 4 to see the sum of all deposits

enter 5 to see the sum of all withdrawals

enter 0 to stop

4

08:04:00 sum of all deposits: 7000

enter 1 to get account balance

enter 2 to deposit money

enter 3 to withdraw money

enter 4 to see the sum of all deposits

enter 5 to see the sum of all withdrawals

enter 0 to stop

5

08:05:00 sum of all withdrawals: 1500

enter 1 to get account balance

enter 2 to deposit money

enter 3 to withdraw money

enter 4 to see the sum of all deposits

enter 5 to see the sum of all withdrawals

enter 0 to stop

0

**בהצלחה!!**