Advanced Programming

Homework Assignment 5

friend, string, Exceptions and static

General guidelines:

a. Maximize readability and ensure indentation.
b. Do exactly as required in the questions.
c. Every class in every question must be submitted in two seperate files – a header file (.h) and an implementation file (.cpp).
d. Add functions and helper methods as necessary to ensure readability.
e. Submission must be done according to the submission guidelines – a document published in the course website – read them carefully!
f. Use relevant names.
g. Use comments to document your functions and complex code parts. **Add an output example at the end of your file!**
h. Individual work and submission – paired submission is not allowed.

Important note: Unless otherwise specified, ever homework has no more than one week for submission.
Open submission boxes past the deadline are not permission for late submission.

Question 1:

Overview

The object of this question is implementation of an ATM system.

In every case of an error throw the correct exception and proceed to the next action. The list of exceptions for each section is detailed below.

For each of the stages below, write code that solves the problem as expected in a state where no exception occurs. Afterwards add exception throwing wherever relevant.
The exception throwing is done in the functions. The exception handling (i.e. catching) is to be done in the main program. **The enclosed main program is to be modified as necessary.**

Reminder: If a rvalue type string is thrown (i.e. a value that can be assigned but not modified, such as a constant string like "Wrong time format"), the catching can only be done in a const char *.

Part A

Define and implement a class called **Clock** that represents the time.

The class must include the following fields (**private**):

- hour – An integer range 0-23.
- minute – An integer range 0-59.
- second – An integer range 0-59.

All the fields should default to 0.

The class must include the following methods (**public**):

- **Assignment Constructor**
  - Receives and assigns values to the fields.
  - If any value is invalid:
    - **All** the fields will remain with default values.
    - A relevant exception will be thrown, according to the following precedence:
      - First the seconds will be tested and an exception thrown if necessary.
      - If the seconds are valid the minutes will be tested and an exception thrown if necessary.
      - If both the seconds and the minutes are valid the hours will be tested and an exception thrown if necessary.

(No copy constructor. Why?)

- **Getters an Setters**
  - For every field.
  - If the value received in the setter is invalid the field is to be left unchanged. A relevant exception is to be thrown.
- **equals** – A boolean method that receives a Clock instance and returns true if the times are identical.
- **before** – A boolean method that receives a Clock instance and returns true if the time in it is later than the time in the calling instance.
- **after** – A boolean method that receives a Clock instance and returns true if the time in it is earlier than the time in the calling instance.
- **operator+=** - receives an int that represents a number of seconds and updates the time accordingly.
- **operator<<** - outputs the time in the format hh:mm:ss (Note: Even if a field is smaller than 10 it should be outputted in 2 digit format).
- **operator>>** - inputs the time in the required format (hh:mm:ss). If an invalid value is entered an exception is to be thrown and the time is to be updated to 00:00:00.

    - The following exception messages are to be used (in this order of precedence):

      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.
    - For any error that these messages do not apply use the following exception message:

      Wrong time format.
    - Any exception message thrown is to be printed to the screen when caught.

<u>Part B</u>

Define and implement a class called **Account** that represents a bank account.

The class must include the following fields (**private**):

- **accountNumber** – the account number.
- **code** – a 4 digit PIN code number (the left digit is not 0).
- **balance** – the amount of money currently in the account.

- **email** (string) – the e-mail address of the owner of the account. A valid address is defined to contain a '@' character, before it a sequence of characters with no spaces and after it a sequence of characters with no spaces terminating with either .com or .co.il.
    - Use the string class defined in the C++ cstring library. Investigate the following link for more information:
      https://www.cplusplus.com/reference/string/string/

The class must include at least the following constructors:

- **empty constructor** – assigns an empty string to email and 0 to the rest of the fields.
- **assignment constructor** – receives a value for each field and assigns them accordingly. If any of the values is invalid:
    - **All the fields** will revert to default values.
    - A relevant exception message will be thrown:
        - For an invalid code: ERROR: code must be of 4 digits!
        - If the code is valid but there is no @ in the email: ERROR: email must contain @!
        - If the code is valid and there is a @ in the email but the email suffix is invalid: ERROR: email must end at .com or .co.il!

The class must include the following methods:

- Getters for all the fields. The fields cannot be changed after construction except through the input operator so setters are not required.
- **operator>>** - inputs initial values for the fields: accountNumber, code and email. The initial balance is always 0. If any of the inputs is invalid then **none of the fields** are to be changed and a relevant exception message is to be thrown:
    - For an invalid code: ERROR: code must be of 4 digits!
    - If the code is valid but there is no @ in the email: ERROR: email must contain @!
    - If the code is valid and there is a @ in the email but the email suffix is invalid: ERROR: email must end at .com or .co.il!
- **withdraw(int)** – a method for withdrawing cash from the account in the sum provided in the argument. A single withdrawal can be up to 2500 NIS, with an overdraft of up to 6000 NIS. The following exceptions can occur (no changes to the account):
    - If the negative balance resulting from the withdrawal is greater than 6000 NIS.
    - If the negative balance is not greater than 6000 NIS but the withdrawal sum is greater than 2500 NIS.
- **deposit(int)** – a method for depositing cash or checks into the account. The sum may not be greater than 10000 NIS. If the sum is greater than 10000 NIS an exception is thrown and the account is not changed.

The class must include the following static fields and methods:

- **sumWithdraw** – a field containing the total number of withdrawal operations done (in all the accounts).
- **sumDeposit** - a field containing the total number of deposit operations done (in all the accounts).
- getSumWithdraw() – returns the total number of withdrawal operations done (in all the accounts).
- getSumDeposit() - returns the total number of deposit operations done (in all the accounts).

The following exception messages must be used where relevant (except where otherwise stated):

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

Given below is a main program for implementing the ATM system. The program makes use of the classes you defined, assuming a bank with no more than 10 accounts.

In the first stage, the user is required to input the account info for all the accounts:

- Account number (must be unique – no two accounts can have the same number).
- Code (4 digits, the left digit not 0).
- Email address (A sequence of characters, no spaces, contains a @ and has one of the suffixes .com or .co.il).

The user does not input the balance. Assume the initial balance is always 0.

At this stage the following exceptions can occur:

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!

In the next stage, the program allows the user to choose operations to perform, one at a time, in a loop. The loop ends when the user choice is 0 (stop).
For each operation, the program performs the operation and prints a relevant message containing the time the operation was performed. In the case of an exception the program is to print the time of the exception and then the exception message, after which the loop is continued to the next user choice.

The program has the following assumptions:

- The ATM starts the day at 08:00:00 (8 AM).
- Checking the balance takes 20 seconds.
- Withdrawal takes 50 seconds.
- Deposit takes 30 seconds.
- Printing the total number of deposits or withdrawals takes a minute.
- All operations are done one after the other with no intervals between them.

Note: The exceptions are thrown from the methods (and functions) and are not in the main function itself. They are only caught in the main function. The exception throwing must be done in the beginning of the relevant methods.
The exceptions must be caught in the main function in the relevant places. Therefore, you are required to modify the main function so that it can catch all the possible exceptions.

(As an example one of the try-catch blocks has already been inserted. Add the others where relevant).

```
#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;
}
```

Output example (a program with 3 accounts and no exceptions):

```
enter account number, code and email for 3 accounts:
123  4444 me@gmail.com
 5555 234 you@walla.co.il
 6666 345 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
```

Good Luck!