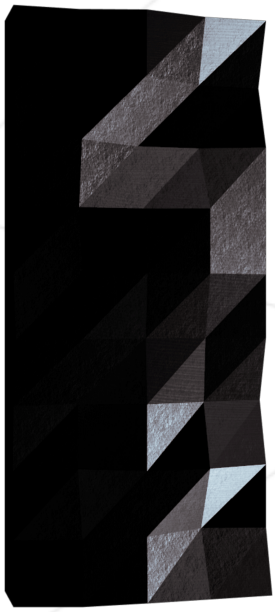

Advanced Object Orientation

Day 00 - Encapsulation

Summary: This document will introduce you to an important part of object oriented programming : Encapsulation

Version: 1.00

Contents

Chapter I

Preamble

In this day, you will be introduce to a concept really important in object oriented programming.
Let's stop for 5 seconds to define a number of names that will serve you in this pool.

- "Object": a concept embodying a "thing".
- "Instance of an object": A concrete entity corresponding to a given object.
- "Attributes" : An internal variable of the instance of an object
- "Method": a way to communicate with an instance of an object, to make it perform actions.

Now that we agree on these 4 words ...

Behold ... You guessed it !... The "Encapsulation" !

Behind this barbaric name hides in reality a rather simple concept: There are things that "belong" to an object, and these things may or may not be accessible by others.

In Object Oriented Programming (OOP), we can define attributes and methods in two ways:

- Private : these attributes/methods will be usable only and strictly only by the object itself
- Public : these attributes/methods can be used/modified by the outside of the class

The idea is to make the object a bit like a black box : the exterior can only access what is public, and nothing of what happens in the black box, which would be private.

```
Yes, it do exist the Protected type, but we will discuss this later
on
```

Chapter II

General rules

- Your program should not crash in any circumstances (even when it runs out of memory), and should not quit unexpectedly.
  If it happens, your project will be considered non-functional and your grade will be `0`.
- You have to turn in a `Makefile` which will compile your source files. It must not relink.
- Your `Makefile` must at least contain the rules:
  `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- Compile your code with `c++` and the flags `-Wall -Wextra -Werror`
- Your code must comply with the **C++ 98 standard**. Then, it should still compile if you add the flag `-std=c++98`.
- Try to always develop using the most `C++` features you can (for example, choose `<cstring>` over `<string.h>`). You are allowed to use `C` functions, but always prefer their `C++` versions if possible.
- Any external library and `Boost` libraries are forbidden.

Chapter III

Exercice 00 : Divide and conquer

	Exercise
Exercice 00 : Divide and conquer	
Turn-in directory : *ex/*	
Files to turn in : `ex00/*.cpp, ex00/*.hpp, ex00/main.cpp, Makefile`	
Allowed functions : `standard STD containers and structure`	

In this exercice, you must edit the code contained inside the DivideAndRule.cpp file. It contain :

- A banking system
- A customer account system

It represent a bank, with a customer account system.

But this code was made by a developer who didn't follow the Advenced Object piscine of 42, and therefore couldn't know how to make correctly encapsulated classes! Everything is accessible by everyone !

So anyone can change the value of his own bank account! Very practical! But not very logical .... Same for loans, everybody can get them without having to do anything, or even having to pay them back!

All this is not very clean ... So....

Your job will be to take this code, and encapsulate correctly the two classes Account and Bank, so that no action can be done illogically !

You must meet at least the following requirements:

- The bank must receive 5% of each money inflow on these client accounts
- The accounts must never have two identical IDs
- The attributes of the structures must not be modifiable from the outside
- The bank must be able to create, delete and modify the accounts of these customers
- The bank must be able to give a loan to a customer, within the limits of its funds
- It must be impossible to add money to a client account without going through the bank
- If it makes sense, the creation of a Getter and a Setter is mandatory. Getter(s) by copy will not be accepted
- If it makes sense, the creation of a const Getter is mandatory. const Getter(s) by copy will not be accepted

During the evaluation, each choice made during this exercise must be defended.

Chapter IV

Exercice 01 : I don't know what i'm doing !

	Exercise 01
Exercice 01 : I don't know what i'm doing !	
Turn-in directory : *ex*01/	
Files to turn in : `ex01/*.cpp, ex01/*.hpp, ex01/main.cpp, Makefile`	
Allowed functions : `standard STD containers and structure`	

In this exercice, you're required to apply what you learned from the last exercice. You must create a contener Vector2, which must contain two components X and Y, of type float. You must create a class Graph, and place some point (represented by the new structure Vector2) onto it.
Once every points added into it, you must output it on screen, via an ascii art

`Think about the encapsulation of those two attributes : do you want them to be private ? public ?`

Create a Graph class, which must contain a size, in Vector2 and a list of Vector2 representing the different points of the graph.

```
You must think of the encapsulation of this class :
- How will you handle adding of a new point ?
- How will you output the graph onto the console ?
```

Here is an example of what is expected as output for a graph composed of the following points :

- 0/0
- 2/2
- 4/2
- 2/4

```
>& 5 . . . . . .
>& 4 . . X . . .
>& 3 . . . . . .
>& 2 . . X . X .
>& 1 . . . . . .
>& 0 X . . . . .
>&   0 1 2 3 4 5
```

You must meet at least the following requirements:

- You must be able to explain the decision you made over encapsulation.
- Vector2 must contain an attribute representing x and y.
- User must be allow to add a new Vector2 onto the Graph
- User must be able to output the content of the graph onto the console in a graph-like form

Chapter V

Bonuses

As bonuses, we propose more requirements for both exercices

V.1 Exercice 00 : Divide and Govern

- You can't create any other methods than const getters in Account structure
- The Account structure must be internal to the Bank structure.
- The Bank structure must contain an operator[] to get an account by it's ID, but you aren't allow to make a while or a for loop to find the account corresponding to such ID
- The error management must be done via throw, and the main must handle those errors

V.2 Exercice 01 : What am i looking at ?!

- Creation of a graphic representation in a PNG of the Graph class, where all the points will be visible.
- Add a line feature to the graph class
- Read an input file, containing a list of point

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter VI

Submission and peer-evaluation

For this evaluation, there will be verification that you have correctly encapsulate the classes, but there will be also questions about why you made thoses changes, and who it's logical to made those changes.

The idea is to force you to THINK of what you do, and not simply follow rules without fully understanding why there are here.

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.