Practicum 6 – The Singleton Pattern

Assignment Details

Assigned: February 25th, 2013. Due: February 28th, 2014 at midnight.

Background

This practicum assignment will introduce students to the singleton design pattern.

The Practicum

So, log in as normal, go to your `la6` directory. We're going to introduce a new concept – the singleton design pattern.

We talked about static methods in class. For this practicum, we're going to work on both a use of static methods, and an additional concept – static variables.

Static variables

Static variables can be declared in two ways; as a member variable of a class, or as a variable declared inside a function/method.

In both cases, there's only *one* copy of the variable in memory, regardless of how many objects ever get created.

This means that you can initialize a variable inside a function, and if it's declared static, it will maintain state across calls to the function. This lets you do some tricks. For starters, let's make a main.cpp in your la6 directory.

In addition to a main function and the necessary `#include`s and `using`s, add a function:

```
void static_test()
{
	static int a = 1;
	cout << a << endl;
	a++;
}
```

Now, call this function three times in your main function. Does it do what you expect it to?

This can be tricky – even though it gets the value 1 assigned to it in its declaration, static variables are only initialized once – and the "a = 1", by virtue of being part of its declaration, is an initialization and will only happen once.

So, now let's create a singleton…

The Singleton

Ok, here we go.

We're going to create a class[1] (and for this assignment, you do not have to use separate compilation (i.e., separate header and .cpp files), but you may if you want).

Declare an integer member variable in the class.

Declare a constructor, but make the constructor private. In the constructor, output something to the console noting that the object has been constructed. Make sure to initialize the integer variable to something reasonable!

Declare a destructor, and in the destructor, output something to the console noting that the object has been destructed.

Declare a public method which sends output to the console that it's been called, and then outputs the integer variable to the console and, finally, increments it.

Declare a static method of the class with a return type of a reference to an object of the class; name this method "instance".

In the instance method, declare a static object of the class (just declare the variable without pointer semantics; while normally this generates an object on the stack, with the static keyword, it actually goes on the heap[2]). Return that object to end the method call.

In your main function, after you call `static_test` (from above), call the output/increment method three times. You'll need to do it by getting a reference to the singleton object via a static method call[3], and then from that reference call the method.

After you call it three times, output something to the console saying your main function is about to exit.

Requirements

Check in both your code for this assignment as well as a test run (e.g., if you compiled the program to a.out, "`./a.out > testrun.txt`" will generate a test run). Answer the following reflection questions in an additional text or PDF file.

---

[1] Part of the point of this exercise is to be able to read the description and generate code from it, hence the lack of code in the description… As always, if you don't understand something, ask.

[2] This is a *significant* oversimplification, and compiler dependent to boot. In short, it's not *exactly* in the heap, but the way most compilers (including g++) store it in memory, it will end up on the heap "side" of things as opposed to the stack side of things. At the least, it will *not* be close to another variable declared on the stack, non-statically, in the same function, and prior to EE/CS380 and CS441, "on the heap" is a *reasonable* synonym for "not on the stack".

[3] Hint: Class name, scope resolution operator, method name.

Reflection Questions

1) Normally, a local variable declared in a function is allocated on the stack, not on the heap[4], but above it says this is not the case if the variable is static. Why can the variable not be allocated on the stack?
2) What does the order of console output from your program tell you about when the static object is initialized?
3) Does your object get deallocated in a reasonable way?
4) How would the output of the program be different if the return type of your instance method was not pass-by-reference? Why?
5) Test #4 by making the change to the class. How many times does your constructor get called? Your destructor?

[4] Note the caveat regarding stack/heap placement of static function variables on the previous page, and then note it again.