CS 215 Program Design, Abstraction, and Problem Solving

Programming Project #5

INTRODUCTION

The goal of this programming project is to give the student practice of writing a C++ std::threads, using the Observer design pattern and experience with locks.

PROJECT TASKS

1. Read the problem definition below and write a C++ program that implements your solution. Readability of the program will be graded based on;
   - a variable naming,
   - b indentation,
   - c **There should be comments at the following points:**
     - i At the beginning of the main to describe the purpose of the program
     - ii In the header files: before every public method and class to describe task, parameters, return values.
     - iii Throughout the code to discuss any complex logic.
   - d spacing,
   - e consistency
   - f If a method overrides a base class method the override key word must be used.
   - g styling in general including choice of methods
2. Compile and test your program
3. Run the code and write test code.
4. All classes will be in separate cpp and h files.

GOALS (or why are we doing this project)

The following are the things to be learned or show you have learned by this project.

- The Observer pattern
- Learning how to use C++ std::threads and locks
- Learn how to have notification pushed to a class
- Designing for testing.

PROBLEM

You are going to enhance programming project #4.
The first enhancement is to make the Singleton class Counter thread safe.
The 2nd enhancement is to add a notification system to the class Counter. This system will notify all registered classes when the value of counter changes. This should be implemented via the Observer pattern.
The 3rd enhancement is to create an observer class that monitors the Counter class to get any changes of Counter's counter.
The 4th enhancement is to remove the sleep from the class Counter.

Requirements

Thread safty

The first enhancement is to make the Singleton class Counter thread safe.

Notification System

The 2nd enhancement is to add a notification system to Counter. This system will notify all registered classes when the value of counter changes. This should be implemented via the Subject/Observer pattern by deriving from Observer (yes this means you need to implement the class Observer).

Observer Class

The 3rd enhancement is to create an observer class that attaches to the Counter class to get any changes of Counter's counter. This class should be derived from Subject (yes this means you need to implement the class Subject). When the Observer class gets notified of a change in Counter's count value it should print it out.

Main

Create a function incCounter to call Counter and increment it 50 times in a loop.
Create two threads one for the incCounter function and one for calling IncCounter::execute passing in 50
When the two threads finish running get the current value of Counter.

Test Class

Note: You must have a CounterTest class and call its methods (1 test per method) to execute the test from main. FYI: The method should return bool (true for success, else false).

TESTING

Write a test class **CounterTest** using the Arrange/Act/Assert design.
Insure that it has the following test cases:

Test 1

Create Counter
Increment Counter
Get the current value of Counter, test for 1.

Test 2

Counter& instance1 = Counter::instance();
Counter& instance2 = Counter::instance();
Counter& instance3 = Counter::instance();
Get current value of Counter

Increment instance1

Increment instance2
Get the current value of Counter (from instance3), test for the value being 2 greater than it was.

Test 3

Get the current value of Counter.
Create an Observer for Counter. // FYI if it was me I'd create an Observer than set a flag when called and set a counter to the value it gets from the Subject.
Increment Counter
Note if your Observer was called. If it was success, else failure.
Note that the value your Observer was called is one more than the value obtained at the beginning of the test, if it is success else failure.

Add at least one of your own test.

SUBMISSION

a) Upload to Blackboard a copy of your source program (the one(s) with a .cpp and .h extensions). (please zip up your files into one submission, *.h, *.cpp, and makefile)

MAXIMUM POSSIBLE SCORE

This program will be graded out of 100 points distributed as follows:

ITEM	MAX. POINTS
Style: see Project Task #1, . . ….....	20
Program written to specification	50
Program works correctly	30

*** Programs that have syntax errors will automatically get a score of 0 ***