CS 216 Introduction Software Engineer
Programming Project #4

INTRODUCTION
The goal of this programming project is to give the student practice of writing a C++ std::threads, using the Singleton design pattern.

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 styling in general including choice of methods
    g If a method overrides a base class method the override key word must be used.
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 Singleton Pattern
- Learning how to use C++ std::threads
- Designing for testing.

PROBLEM
You are going to write a program using std::threads. Run the finished program ten or more times noting the number printed out. It should be 100, is it? Turn in the number you observed.

Requirements
Write a Class (lets call it Counter) using the Singleton pattern that keeps track of a integer counter.
- The counter starts at zero and is incremented via a call to a method in the Singleton class, in this method after incrementing the counter sleep for 10 milisecountds. (`#include <thread>`)
```
std::this_thread::sleep_for(std::chrono::milliseconds(10));
```
- The class also has a method to let you get the current value of the counter (lets call the method `int get() const;`)

IncCounter
Create a class IncCounter with a method *execute* that will call Counter and increment it N times in a loop where N is a parameter of *execute*.

main

Create a function incCounter (in main.cpp) 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 and print it out.
Run the program at least 15 times, noting the final value of Counter each time.

TESTING

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

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

```
Counter& instance1 = Counter::instance();
Counter& instance2 = Counter::instance();
Counter& instance3 = Counter::instance();
```

Increment instance1
Increment instance2
Get the current value of Counter (from instance3), test for 3.

Add some 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 cmakefile)
b) Note in some manner the final value from your 15 runs.

MAXIMUM POSSIBLE SCORE

This program will be graded out of 100 points distributed as follows:
If a project works but does not have the code to satisfy the core objective, has syntax errors, or is its classes are not separated into definition and implementation files it will automatically be awarded a score of 0.

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 ***