A SIMPLE READER/WRITER APPLICATION

1 OBJECTIVES

The main goals of this assignment are:

- Handling communication between threads using a simple writer/reader pattern.
- Observing the behavior of non-synchronized threads.
- Learning how multiple threads can be synchronized when accessing a shared resource.

2 DESCRIPTION

Implement a Writer/Reader pattern to solve this assignment, as illustrated in the following simple class diagram.

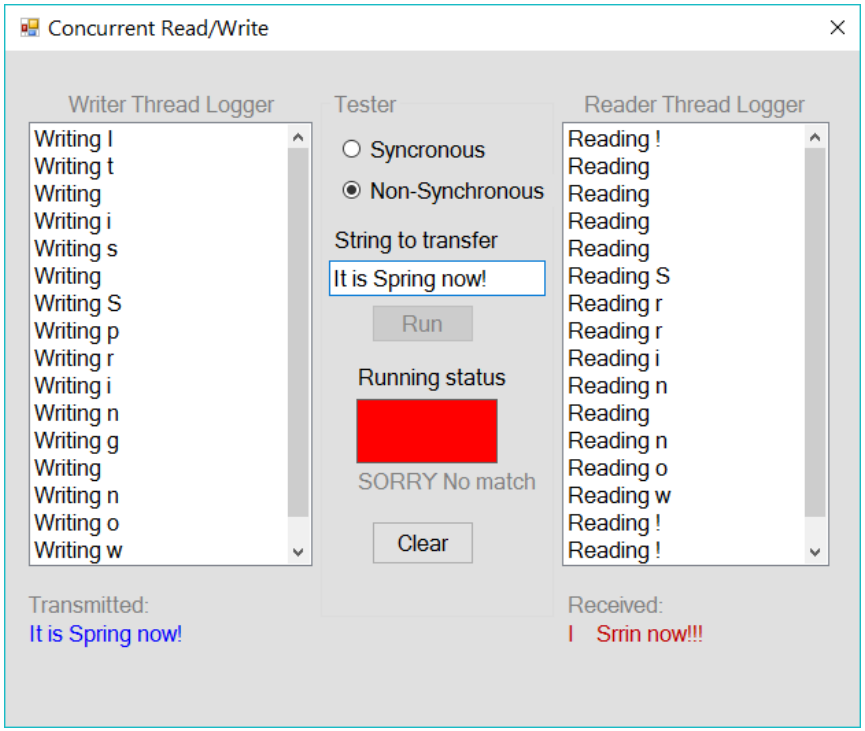
We are going to use the technique with a writer class, a reader class and a third class, the character buffer as the shared object. The GUIs for both Java and C# is provided. If you want to design your own GUI, you don't have to use the given code. You may also change the GUI or use only parts of them. The GUIs are provided as extra help so you can concentrated on the multithreading programming.

For both writing and reading, we are going to use threads. The writer thread puts a character from a text source to the buffer while the reader reads from the buffer.

We test the process in two modes: a) Synchronous where you can use a synchronization mechanism (for example **synchronized** in Java and a **Mutex** or **lock** in C#), and b) Non-synchronous in which no synchronization mechanism is applied.

3 CHARACTER TRANSFER

The application should let the user input a string for use as the text to be transferred from the input source (a text-field on the GUI) to the target source (a ListBox on the GUI). The writer thread writes the characters of the string (one by one) to the shared buffer as long as the buffer is not full. The reader thread reads the characters (one by one) from the shared buffer to a target as long as there are characters to read and the buffer is not empty.



The following requirement are for the assignment:

- The character buffer should hold only ONE character.
- The buffer also has a bool variable like “HasCharacter”.
- NOTE! This bool value should not be tested by a property, it should ONLY be tested Inside the critical section.
- In the Non-synchronous mode, only simple get/set should be done.
- All synchronization takes place in buffer, NOT in writer/reader.

- The reader should put all the read characters back into another string.
- NO **sleep** in buffer.

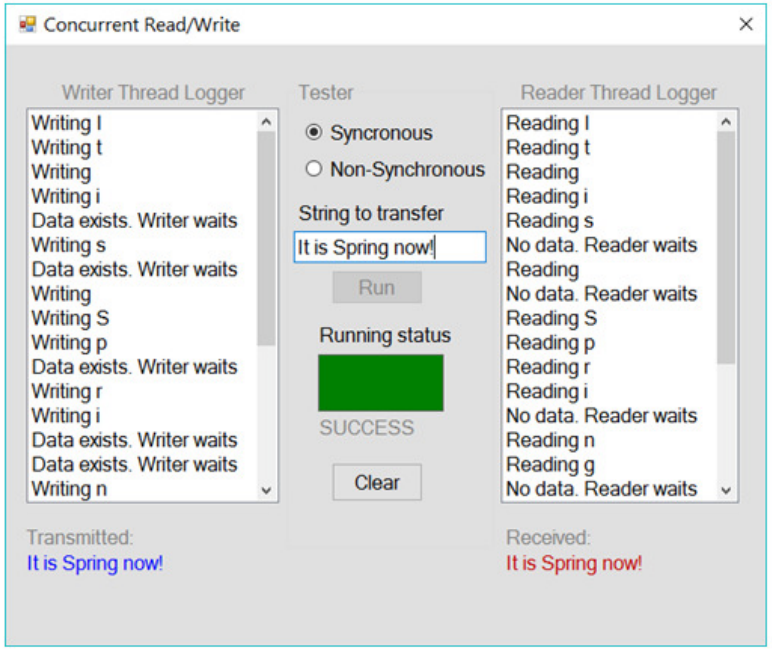
The transfer should be done character by character with a random waiting interval between each character writing and character reading, in order to create a timespan long enough to watch each step on the output window; otherwise things will happen too quickly. The states of the process should be displayed to the screen for the user, as demonstrated in the run sample images (Demo 1a: and 1b). The user should be given the option of choosing to run the application in both synchronized and non-synchronized modes. You may use any development tools, and IDEs as well as any of the languages, C#, Jave or C++. Send the string itself to the writer (for looping through its characters) and send its Length to the reader (it should loop reading the same number of characters).

3.1 The GUI based application

3.1.1 Create a GUI-based application and design the user interface with necessary input/out components so the user is able to input a text string and run the application, with either synchronized or non-synchronized option. The user should also see a log of the readings and writings as they happen.

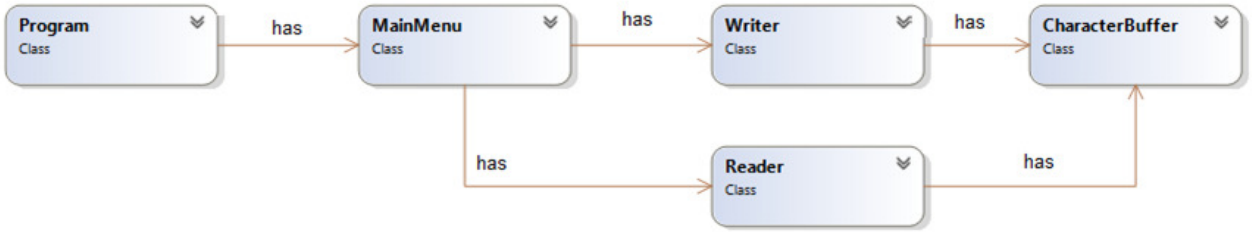
3.1.2 Create and initialize an instance of the above-mentioned classes in the user-interface class (MainForm in C#) or another start class.

3.1.3 Do the necessary programming to accomplish the task (see the run examples images to get an idea of how the application is expected to work).



3.2 Console based application

3.2.1 If you find GUI-based programming (in particular in C++) difficult, you may implement a console application. In this case you use a program structure as below can be applied.



4 SPECIFICATIONS AND REQUIREMENTS FOR A PASS GRADE (G)

4.1 To qualify for a pass-grade, you should implement at least one of the above alternatives with good code quality.

4.2 Do your programming work well structured, well organized, and always have OOP in mind. Use proper variable, method names, and document your code by writing comments in your code.

4.3 NO logic in GUI classes! And NO GUI handling in the logic classes. This implies that the GUI class should only handle the interactions between the user and the program,

4.3.1 reading input from the GUI components,

4.3.2 saving them in related objects,

4.3.3 getting output from the related classes.

4.4 Comment your files with xml headers on classes and methods (C#), comment with JavaDoc in java files.

4.5 You may certainly bring changes in the application to make it more it more fun and full-featured.

4.6 Test your application carefully before submitting.

5 GRADING AND SUBMISSION

Compress all the files, folders and subfolders into a **zip**, **rar, 7z** file, and then upload it via the Assignment page on Canvas. Click the Submit-button "Submit Answer" and attach your file. Do not send your project via mail! Make sure that you submit the correct version of your project and that you have compiled and tested your project before handing in. Be careful not to use any hard-coded file paths (for example path to an image file on your C-drive) in your source code. It will not work on other computers. Projects that do not compile and run correctly, or is done with poor code quality, will be returned for completion and resubmission.

After or before submitting your assignment to the module, show your assignment to your lab leader during the scheduled hours in the labs.

6 TIPS AND RECOMMENDED LINKS

C#: In general, avoid locking on a public type, e.g. `lock(this),` or instances beyond your code's control. Best practice is to define a private object to lock on, or a private static object variable to protect data common to all instances.

```
private Object lockObj = new Object(); //instance variable

//lock block
lock (lockObj)
{
      //code    }
```

As we have not yet covered some features that make the work easier for this assignment, here is some guidelines:

To notify a thread in the waiting queue of a change in the locked object's state, the **Monitor.Pulse** in C# and **threadName.notify** in Java can be used. In addition, the following is copied from the Internet:

C#: The equivalent functionality (including the normal locking) is in the Monitor class (foo is the thread object):

```
foo.notify() => Monitor.Pulse(foo)
foo.notifyAll() => Monitor.PulseAll(foo)
foo.wait() =>  Monitor.Wait(foo)
```

The lock statement in C# is equivalent to calling **Monitor.Enter** and **Monitor.Exit** with an appropriate try/finally block. The lock statement is recommended.

Besides the lock, if you don't feel good with monitors, it IS ok to use a bool variable as the result, but remember you can't test it by itself, it must be tested IN buffer.

C++
Where you would have called `java.lang.Object.wait`, call `pthread_cond_wait` or `pthread_cond_timedwait`.
Where you would have called `java.lang.Object.notify`, call `pthread_cond_signal`.
Where you would have called `java.lang.Object.notifyAll`, call `pthread_cond_broadcast`

Try to use names for your components and variables that describes their behavior.

Useful Links:
C#: http://www.yoda.arachsys.com/csharp/threads/
Reader/Writer: http://msdn.microsoft.com/en-us/library/system.threading.readerwriterlock(v=vs.110).aspx
Lock: http://msdn.microsoft.com/en-us/library/c5kehkcz.aspx

Java: http://www.journaldev.com/1037/java-thread-wait-notify-and-notifyall-example
Reader/Writer: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReadWriteLock.html

C++: http://stackoverflow.com/questions/2085511/wait-and-notify-in-c-c-shared-memory
Reader/Writer: http://www.codeproject.com/Articles/598695/Cplusplus-threads-locks-and-condition-variables

Good Luck!

Farid Naisan,
Course Responsible and Instructor