MALMÖ HÖGSKOLA

Concurrent Programming

Assignment 2

Thread Synchronization – A Simple Reader and Writer

Mandatory

University Lecturer
Faculty of Technology and Society

A Simple Rader/Writer Application

1 Objectives

The main goals of this assignment are:

- Provide communication between threads using a simple reader-writer pattern.
- Observe the behaviour of non-synchronized threads.
- Learn how multiple threads are synchronized when accessing a shared resource.

2 Description

In this assignment you are to implement a reader-writer pattern to let two threads communicate through a shared buffer. The assignment can be done either as a Console application or as a GUI based application. The latter may be a little more challenging but more interesting, of course.

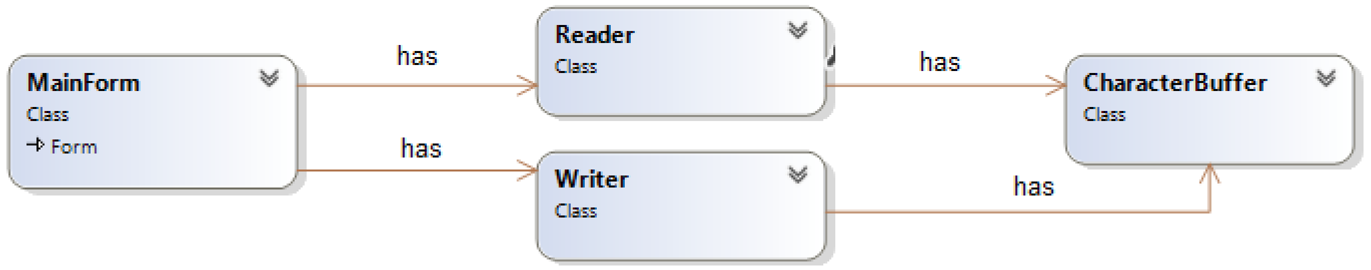
The application should let the user input a string to be used for transferring by a writer to a reader thread using a character buffer as shared resource. The transfer should be done character by character with a random waiting interval between each character writing and character reading, in order to make a timespan long enough to watch each step on the output window; otherwise things will happen too quickly. The states of the application should be displayed to the screen for the user, as demonstrated in the run sample images presented later in this document. The user should be given the option of choosing to run the application in both synchronized and asynchronized modes. You may use any development tools and IDEs and any of the languages, C#, Jave or C++; console or GUI based is optional.

Note: By asynchronized, here in this assignment, it is simply meant not-syncrhonized. Asynchronized programming will be discussed in its technical meaning later in this course.

2.1 The GUI-based application

2.1.1 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 synchronized or asynchronized options. The user should also see a log of the readings and writings as they happen.

2.1.2 Write three classes, **CharacterBuffer**, **Writer** and **Reader**, associated to each other as shown in the class diagram below.



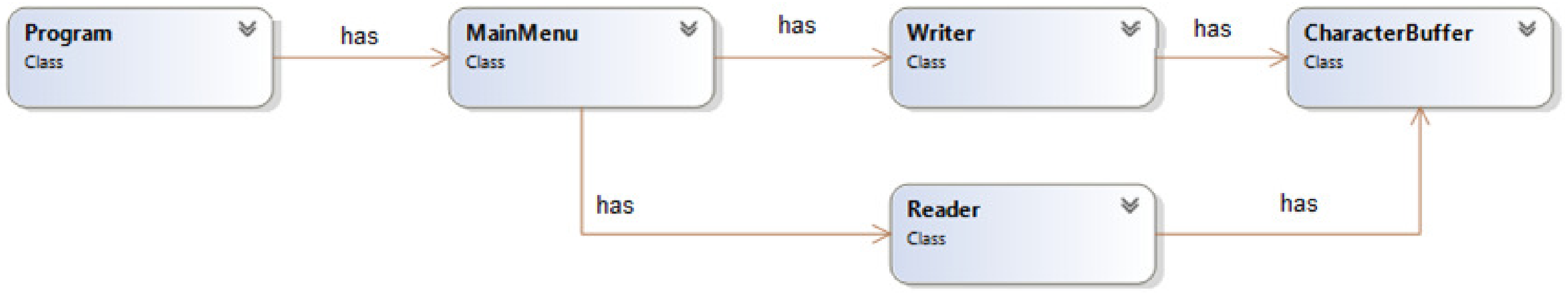
2.1.3 Create and initialize an instance of the above-mentioned classes in the **MainForm**

2.1.4 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).

2.2 Console based application

2.2.1 Let the user input a text to be placed in the buffer for testing, and ask the user whether to apply synchronization or not.

2.2.2 Create your classes as shown in the class diagram below.



2.2.3 The **Program** class contains the **Main** method which creates an instance of the **MainMenu**. A C# version may contain code like this:

2.2.4 The Start of the Menu class has call to other methods to run the application and perform the task.

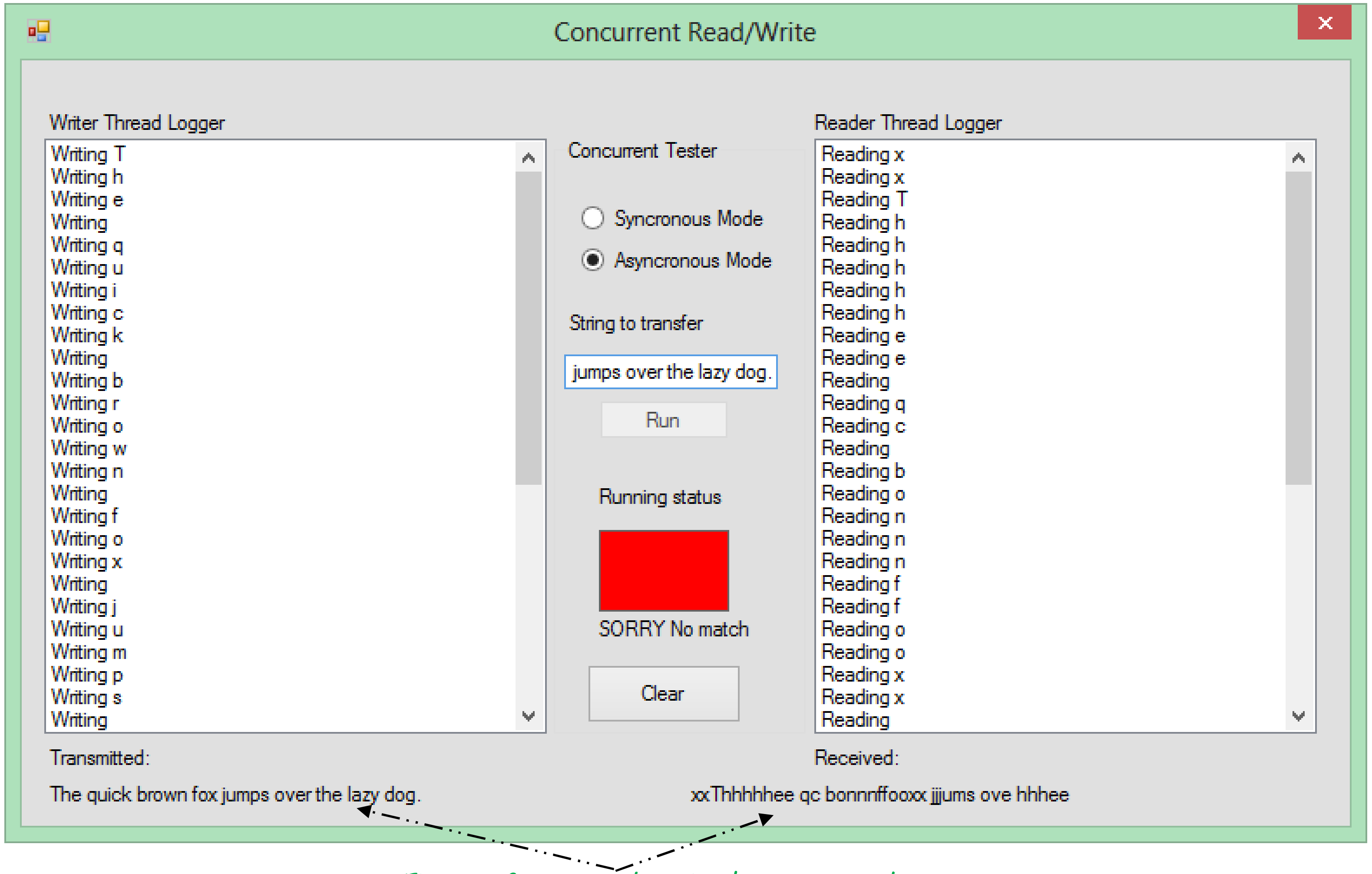
```
namespace ConcurrentReadWrite
{
    class Program
    {
        static void Main(string[] args)
        {
            //Console window properties
            Console.Title = "Reader/Writer Thread Synchronization";
            Console.BackgroundColor = ConsoleColor.White;
            Console.ForegroundColor = ConsoleColor.DarkBlue;
            Console.Clear(); //paint the screen with the background color

            //The actions starts in the Start method of the Menu class
            MainMenu menu = new MainMenu();
            menu.Start();

            //Let the console window remain on th screen when everything is done.
            Console.ReadLine();
        }
    }
}
```

2.3 Run example – GUI based application

The following two images are screen dumps from two run sessions showing asynchronized and synchronized results. The images come from an application created using Visual Studio and C#.



Syncronous Mode [✓ Asyncronous Mode]

```
Writing T
Writing h
Writing e
Writing
Writing q
Writing u
Writing i
Writing c
Writing k
Writing
Writing b
Writing r
Writing o
Writing w
Writing n
Writing
Writing f
Writing o
Writing x
Writing
Writing j
Writing u
Writing m
Writing p
Writing s
Writing
```

```
Reading x
Reading x
Reading T
Reading h
Reading h
Reading h
Reading h
Reading h
Reading e
Reading e
Reading
Reading q
Reading c
Reading
Reading b
Reading o
Reading n
Reading n
Reading n
Reading f
Reading f
Reading o
Reading o
Reading x
Reading x
Reading
```

SORRY No match

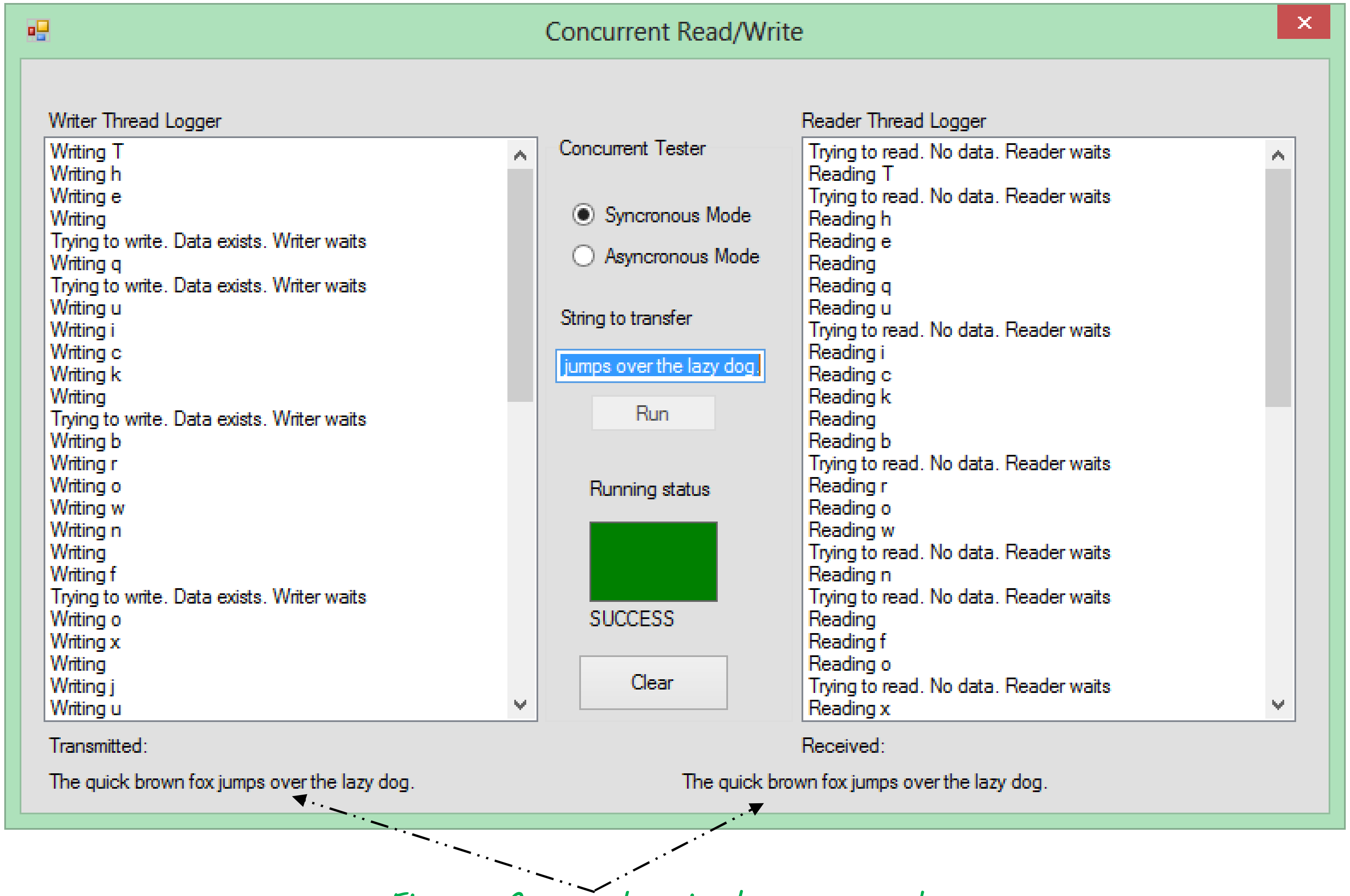
Transmitted:

The quick brown fox jumps over the lazy dog.

Received:

xxThhhhhee qc bonnnffooxx jjjums ove hhhee

Figure: An asnynchronized run example



Concurrent Read/Write

Writer Thread Logger

```
Writing T
Writing h
Writing e
Writing
Trying to write. Data exists. Writer waits
Writing q
Trying to write. Data exists. Writer waits
Writing u
Writing i
Writing c
Writing k
Writing
Trying to write. Data exists. Writer waits
Writing b
Writing r
Writing o
Writing w
Writing n
Writing
Writing f
Trying to write. Data exists. Writer waits
Writing o
Writing x
Writing
Writing j
Writing u
```

Reader Thread Logger

```
Trying to read. No data. Reader waits
Reading T
Trying to read. No data. Reader waits
Reading h
Reading e
Reading
Reading q
Reading u
Trying to read. No data. Reader waits
Reading i
Reading c
Reading k
Reading
Reading b
Trying to read. No data. Reader waits
Reading r
Reading o
Reading w
Trying to read. No data. Reader waits
Reading n
Trying to read. No data. Reader waits
Reading
Reading f
Reading o
Trying to read. No data. Reader waits
Reading x
```

Concurrent Tester

[● Syncronous Mode] ○ Asyncronous Mode

String to transfer: jumps over the lazy dog.

Running status: SUCCESS

Transmitted:

The quick brown fox jumps over the lazy dog.

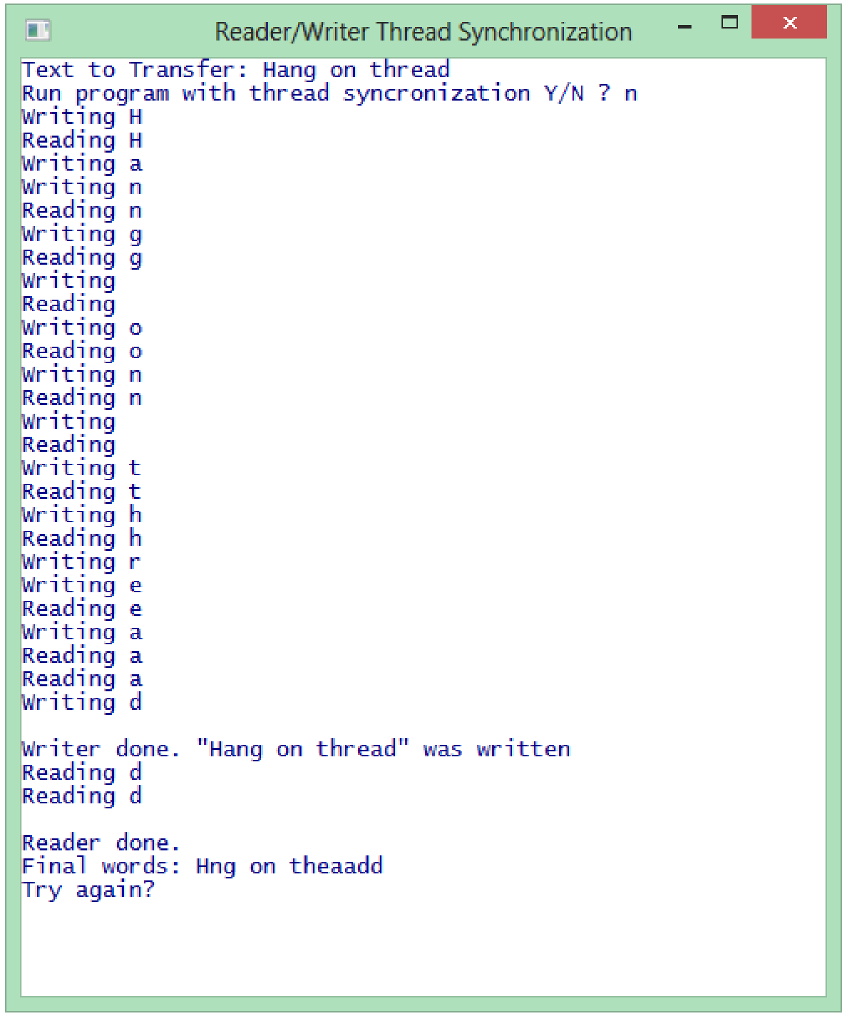
Received:

The quick brown fox jumps over the lazy dog.

Figure: A synchronized run example

2.4 Run example – Console based application.

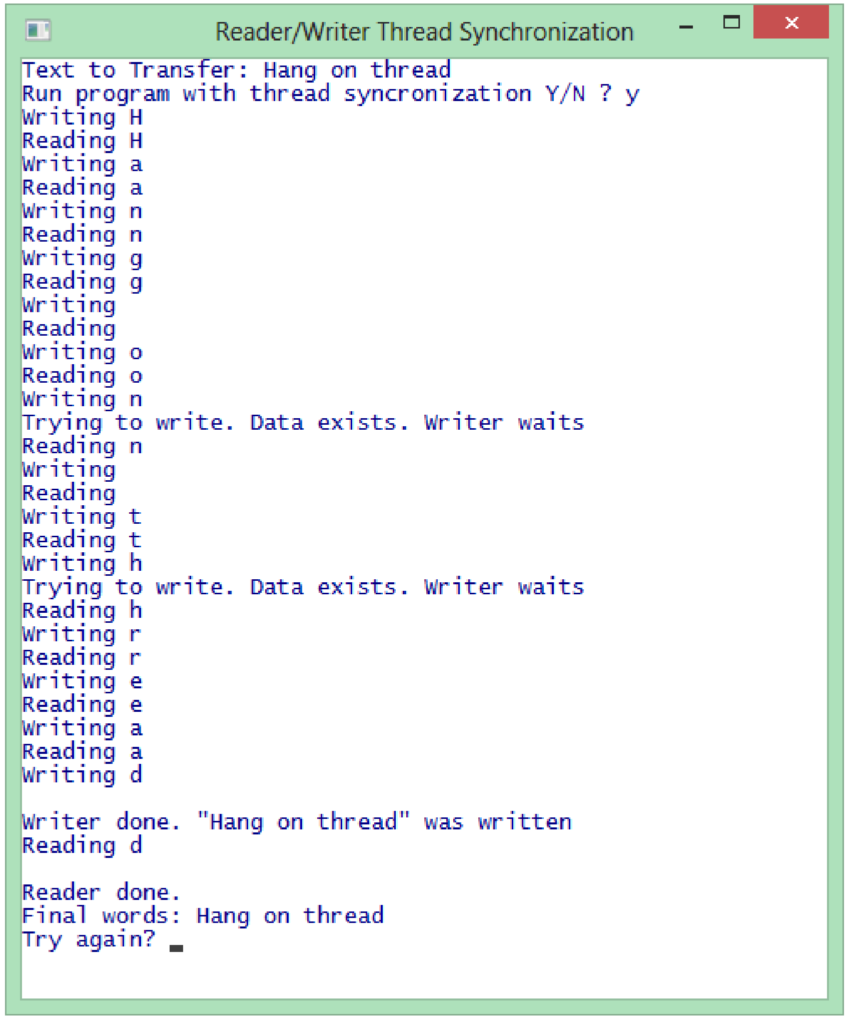
The figures below shows two sample runs of a Console application created using a Visual Studio Console project, and C#. The left picture is a run example without synchronization of the reader and writer threads while the image at the right shows the same but with synchronization of the threads.



```
Text to Transfer: Hang on thread
Run program with thread syncronization Y/N ? n
Writing H
Reading H
Writing a
Writing n
Reading n
Writing g
Reading g
Writing
Reading
Writing o
Reading o
Writing n
Reading n
Writing
Reading
Writing t
Reading t
Writing h
Reading h
Writing r
Writing e
Reading e
Writing a
Reading a
Reading a
Writing d

Writer done. "Hang on thread" was written
Reading d
Reading d

Reader done.
Final words: Hng on theaadd
Try again?
```



```
Text to Transfer: Hang on thread
Run program with thread syncronization Y/N ? y
Writing H
Reading H
Writing a
Reading a
Writing n
Reading n
Writing g
Reading g
Writing
Reading
Writing o
Reading o
Writing n
Trying to write. Data exists. Writer waits
Reading n
Writing
Reading
Writing t
Reading t
Writing h
Trying to write. Data exists. Writer waits
Reading h
Writing r
Reading r
Writing e
Reading e
Writing a
Reading a
Writing d

Writer done. "Hang on thread" was written
Reading d

Reader done.
Final words: Hang on thread
Try again?
```

3 Specifications and Requirements for a pass grade (G)

3.1 Finish the assignment according the above description. You may of course add more features or use more threads to improve the application.

3.2 Code quality is important. Put some effort on the structure and documenting your code. Create at least as many classes as shown in the class diagrams. The classes should have methods for every separated task.

3.3 Test your application carefully before submitting.

4 Grading and submission

Show your assignment to your lab leader during the scheduled hours in the labs, but before doing so, you must upload your work to Its' L. 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 will be directly returned for completion and resubmission.

Compress all the files, folders and subfolders into a zip, 7z or rar file, and then upload it via the Assignment page on It's L. Click the button "Submit Answer" and attach your file. Do not send your project via mail!

5 Tips and links

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.

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`

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