ConcatStream and Tests
Cpt S 422 Homework Assignment
by Evan Olds

Assignment Instructions:

Read all instructions *carefully* before you write any code.

In this assignment you implement the ConcatStream that provides the capability to combine two streams into one. You will also implement unit tests that will be submitted with your assignment zip file. One additional stream must be designed to assist with these tests and it is described on page 2.

Implement the ConcatStream Class

Implement the ConcatStream class in the CS422 namespace. Implement it in the ConcatStream.cs file. Remember to match the naming exactly, including casing. This is the class we discussed in lecture, that inherits from Stream and concatenates two streams together into one.

Important: even though the web server will only use the ConcatStream as a read-only stream, you must implement both the Read and Write functions. Each should throw an exception if either of the 2 streams does not support that action. The writing functionality will be described later.

The overall functionality of the ConcatStream is limited by the lesser-functional of the two streams being concatenated. So the ConcatStream can only seek if both streams can seek, can only read if both streams can read, and so on. Implement two constructors in this class:

```
ConcatStream(Stream first, Stream second)
ConcatStream(Stream first, Stream second, long fixedLength)
```

The first constructor takes two Stream objects, and doesn't need to support the "Length" property unless both streams support it. However, it's only the second stream that is permitted to not have a length. If the first of the two does not have a length, then it's not possible to know where it ends and the next stream begins, so in this case throw an ArgumentException. The ConcatStream cannot be implemented without knowing the length of the first stream.

The second stream, on the other hand, should be allowed to be a stream without a length or the capability to seek, such as a NetworkStream. If the second stream doesn't support seeking, have the ConcatStream provide forward-only reading functionality with no seeking.

Much of the above applies to the second constructor as well, only this time even if the second stream does not have a length, you will support the Length property. Have the second constructor store the fixed length value and return it in the getter for the Length property. You'll also need some way of

storing information about which of the two constructors was used to instantiate the class, since the Length property behaves differently based on how the ConcatStream is constructed.

The writing functionality will be such that it never expands the length of the first stream. As an example, suppose you have 2 writable streams, the first of which is 20 bytes in size and currently at position 10 (meaning the ConcatStream's position is also at 10), and you write 30 bytes from a buffer. Since there are only 10 bytes left in the first stream and you cannot expand that stream, you write the first 10 of 30 bytes to it. Then you write the remaining 20 to the second stream.

If a write call cannot be completed then throw an exception, but only do so if the action cannot be completed. For example, if your ConcatStream position is 32 and the first stream has a length of 20, then you could only write correctly to the second stream if its current position was exactly 12. In the case when the second stream supports seeking, it's easy enough to ensure that, so just seek to the appropriate spot and write in those circumstances. But if the second stream doesn't support seeking, then only complete the write if you are at the exact correct position. Otherwise throw an exception.

Implement the NoSeekMemoryStream Class and Tests

Since you'll want to test the ConcatStream with a second stream that does not support seeking, you will need such a stream to use in unit tests. Network streams are good, since they are ultimately what we will be using as the second stream in our web server, but they are excessive for a set of simple unit tests. Therefore, you will implement the NoSeekMemoryStream that behaves just like a memory stream, but does not support seeking.

Implement the NoSeekMemoryStream in the CS422 namespace and in the NoSeekMemoryStream.cs file. Have it inherit from MemoryStream and start with the declaration below, filling in the missing pieces as needed.

```
/// <summary>
/// Represents a memory stream that does not support seeking, but otherwise has
/// functionality identical to the MemoryStream class.
/// </summary>
public class NoSeekMemoryStream : MemoryStream
{
    public NoSeekMemoryStream(byte[] buffer) // implement

    public NoSeekMemoryStream(byte[] buffer, int offset, int count) // implement

    // Override necessary properties and methods to ensure that this stream functions
    // just like the MemoryStream class, but throws a NotSupportedException when seeking
    // is attempted (you'll have to override more than just the Seek function!)
}
```

Write a set of unit tests for your ConcatStream class. Include some unit tests that use a NoSeekMemoryStream as the second stream and make sure that your ConcatStream supports forward reading through all the stream contents.

Write additional unit tests as you see fit for ConcatStream as well as a few for the NoSeekMemoryStream class itself. Include the following and remember to submit all unit test code as well as the ConcatStream and NoSeekMemoryStream code:

- ConcatStream unit test that combines two memory streams, reads back all the data in random chunk sizes, and verifies it against the original data
- ConcatStream unit test that combines a memory stream as the first and a NoSeekMemoryStream as the second, and verifies that all data can be read
- ConcatStream tests that query Length property in both circumstances where it can and cannot be computed without exception.
- NoSeekMemoryStream unit test that attempts to seek using all relevant properties/methods that provide seeking capabilities in a stream, and makes sure that each throws the NotSupportedException.

Your unit tests can either be NUnit-based tests that run in MonoDevelop or standalone terminal application. Implement unit tests in different code files than the stream classes.