Streams

BY EVAN OLDS

CPT S 422

Stream

- Abstract data type
- Conceptually represents a linear sequence of bytes
- System.IO.Stream abstract base class
- Functions / Properties:
  - Read - function, reads data from the stream
  - Write - function, writes data to the stream
  - Position – integer property that gets the current position of the stream (if supported). In streams that support seeking this can also be set in order to seek.
- The read and write functions start reading from and writing to the stream at the location specified by Position
- Remember that if a function takes a stream object as a parameter then anything that inherits from stream can be passed to that function.

Stream Functions/Properties

- Read - function, reads data from the stream
- Write - function, writes data to the stream
- Position – integer property that gets the current position of the stream (if supported). In streams that support seeking this can also be set in order to seek.
- The read and write functions start reading from and writing to the stream at the location specified by Position
- Remember that if a function takes a stream object as a parameter then anything that inherits from stream can be passed to that function.

Example: File Stream

- The FileStream class in the System.IO namespace inherits from Stream and can be used to read from and write to files:

```
FileStream fs = new FileStream(fileName, FileMode.Create, FileAccess.Write);
byte[] helloWorldBytes = Encoding.UTF8.GetBytes("Hello World!");
fs.Write(helloWorldBytes, 0, helloWorldBytes.Length);
fs.Dispose(); // Why do we have to do this in a managed language?
fs = null;
```

Example: File Stream

- In the .NET framework, FileStream is already available for you to use
- It inherits from the Stream abstract base class
- The FileStream constructor has several different overloads, giving you different options for opening the file
- Whether or not you'll be able to successfully make a Read or Write call is dependent on if you opened the file for reading, writing, or both
  - The are CanRead and CanWrite properties in Stream

More Stream Methods

- As the abstract data type, Read, Write and Position really define the core aspects of a stream
- But there are several other abstract properties
    - CanRead
    - CanWrite
    - CanSeek
    - Length
    - (a few others that we wont discuss)
- When you have a Stream object, you can use these properties to query capabilities of the stream
- Certain types of streams may not allow seeking

Some Streams in .NET

- FileStream class (System.IO) - Represents a file on disk
  - CanSeek will most likely always be true
  - CanRead / CanWrite depend on how you opened the file
- NetworkStream class (System.Net.Sockets) - Represents communication over a network connection (which could potentially be a connection to a server for retrieving a web page, or to another player in a multiplayer game, and so on)
  - CanSeek will likely be false. If data is being sent over the network you can't "back up" or "jump forward". It's just data flowing in one direction (analogous to water in a stream, which is where the name comes from).
- MemoryStream class (System.IO) – Represents a memory buffer (array)
  - Wraps around an array in memory, so CanSeek should always be true
- GZipStream class (System.IO) - Represents a compressed stream
- CryptoStream class (System.Security.Cryptography) – represents an encrypted stream

New Streams Concepts for 422

- Previous slides were all from prerequisite course
- Let’s consider a scenario where code using streams needs to be tested
- Code in your company’s product (let’s say you’re building a text editor for simplicity) looks something like this:
- bool Load(Stream s) { … }
- How to test that function?
  - Where do we even start?
  - Let’s first discuss some basic testing terms/concepts, then come back to this problem

Basic Testing Terms

- Test Case
  - Set of inputs, conditions, and expected results
- Test
  - The act of exercising software with test cases
- Black-box testing
  - Testing software WITHOUT knowledge of its internal workings/implementation
- White-box testing (sometimes “glass box”)
  - Testing software WITH knowledge of its internal workings/implementation

Important Questions

- True or false? If a program crashes on a particular line of code then that line is incorrect
  - How to prove if true?
  - What's a counterexample if false?
- True or false? Suppose your software was crashing under a certain test case. Then you make a change to the software that prevents it from crashing for this test case. You have fixed the bug.
  - How to prove if true?
  - What's a counterexample if false?

More Testing Terms

- So if we can crash on a line of code that isn't actually "wrong", then what's going on?
- Consider the following definitions to help answer this
- **Fault / bug** – An incorrect step, process, or data definition in a computer program
- **Failure** – An incorrect result; The manifestation of the fault (i.e. crash, gibberish output)

Testing Defined Formally

- The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component (IEEE/ANSI STD 610.12-1990)
- The process of analyzing a software item to detect the difference between existing and required conditions (that is, BUGS) and to evaluate the features of the software items (IEEE/ANSI STD 829-1983)
- Testing can only detect the presence of defects it cannot prove that an application is free of defects

Common Failures in C# Applications

- Exceptions
  - InvalidOperationException
  - ArgumentException, ArgumentNullException, ArgumentOutOfRangeException
  - NullReferenceException, IndexOutOfRangeException, AccessViolationException
  - StackOverflowException
  - OutOfMemoryException
  - others…

Common Failures in C# Applications

- When an app crashes in C# you generally get details about the exception (at a minimum the type of exception) in the console output
- So are there any other types of failures in C# applications that won’t result in exceptions?

Common Failures in C# Applications

- So are there any other types of failures in C# applications that won't result in exceptions? Yes, quite a few. In fact, the MAJORITY of failures most likely don't result in a crash/exception.
- Recall that an app doesn't have to actually crash to produce a failure
  - A failure is "an incorrect result"
  - If an app needs to display the result of 2*2 it may do so without crashing, but if it displays 5 then that's certainly a failure

Testing

- Testing efforts should look to find failures of all types
  - Failures leading to crashes
  - Failures leading to incorrect output
  - Failures leading to insufficient speed/responsiveness of the application
  - Failures that occur only with specific input
- From that last bullet point, we can see that we need to be able to define valid vs invalid input
- App should obviously produce valid output and not fail with valid input, but does it have license to fail when invalid input is given?

Testing

- Robust software will not fail when given invalid input
- Some of this is just the intricacies of the notion of a failure
- Suppose an app tries to open a file that the user doesn’t have access to on the file system
    - If it simply displays a message to the user saying that they don’t have access to that file, and then continues working normally, it’s most likely not considered a failure
    - If it crashes it’s definitely a failure (obviously)
    - If it displays a message like “can’t open file” that may or may not be a failure according to the software development team. Why?

Defining Failures

- Software engineers must be able to define what is or is not acceptable as a “result” from their software.
- There are certain things that easily fit into the yes or no categories, others are somewhat dependent on how good the engineers want the software to be
  - A good company may consider “can’t open file” to be a failure, if they expect to have additional details presented to the user about the nature of the failure

Where to start testing?

- Multiple philosophies / approaches, but you are urged you to consider: design the software with testing in mind
  - In this sense, testing efforts start while you're writing the code
  - Points out that testing is NOT just an "after-you've-written-it" activity
  - In many cases (not ALL cases though) software that is easier to extend is also easier to test
  - We'll cover this more later, it will be a big part of this class
- Let's go back to our specific stream-oriented problem, where we have a function that loads data from a stream
  - `bool Load(Stream s) { … }`

What Could Go Wrong?

- Forget all higher-level things outside the function so we can just start simply
- If a load function needs to load data from a stream, what kinds of tests would you want in place?
- Let’s get the easy ones out of the way: we want a set of tests that provide valid input data streams and verify that the data was loaded correctly?
- Make a list in class: what could go wrong with a function that loads from a stream?

“Large” Data Streams

- The list produced in class hopefully covered many different possibilities.
- Let’s focus specifically on one (which may or may not have come up on the list): large data streams
- If the app loads valid data from a stream 5 megabytes (MB) in size, that does NOT mean that it will handle valid data that’s 5 gigabytes (GB) in size
- Where do we get the larger test streams?

“Large” Data Streams

- Could certainly have a 5 MB test file associated with the set of tests for your software, but do you want to have to generate a 5 GB file and distribute it with your test suite?
- Suppose that your company generates a set of automated tests and then deploys them across a server farm so that the tests can run on a variety of different types of machines (not uncommon in industry).
- What if you had 10 different instances of various ~5GB test input data? Deploy 50GB to each machine just for this category of tests?
- Here’s the scenario that really introduces the problem: Suppose the software by definition must load correctly on machines that don’t even have hard drives.
    - Where to get the test file (discuss in class)?

“Large” Data Streams

- You could probably point a hard drive-free machine to a network drive, but we can get similar effects in many scenarios without having the 5 GB test file at all
- We’ll still have a 5 GB stream passed to the function for testing, but the 5 GB never will exist as a file
- On many machines 5 GB memory allocations fail (even if the machine has more than that much RAM), so it will never exist in memory either
- Where is it then? What would have to be true of a 5 GB stream that isn’t backed by a file or memory?
    - Discuss in class

Procedurally Generated Data

- You can write your own class that inherits from Stream and generates the data dynamically during the calls to Read
- Can represent multiple zettabytes of data with such a stream if we want to
- It never has to have the data in memory or on the disk
- Simple example: every time a read call is made for N bytes, the stream fills N random bytes into the output buffer

Procedurally Generated Data

- In the Read function for our procedural-data-generating stream:

```
public override int Read(byte[] buffer, int offset, int
count)
{
        Random rand = new Random();
        for (int i = 0; i < count; i++)
        {
            buffer[i] = (byte)rand.Next(0, 256);
        }
        return count;
}
```

- Would allow for unlimited-size test streams, so we're getting closer to how this is done, but what's wrong with it (at least 2 major bugs)?

Procedurally Generated Data

- Problems with previous example code:
  - Didn't use the offset parameter. Data was potentially written to the wrong location in the buffer.
  - Data is not consistent. Any byte value at position X has a value when it's read, then if you seek back and read and position X again you'd likely get a different value.
- Is the inconsistency ever ok? What would be the expectation for a stream that you're loading data from?
  - Discuss in class

Consistent, Procedurally Generated Data Stream

- Suppose we want to write a class that inherits from Stream, and represents potentially very large amounts of consistent data, but never stores all the stream data in memory or in a file
- For testing purposes, a stream that just generates a byte value based on that byte's index within the stream should suffice
- So the task is to write a stream that can represent an arbitrary number of bytes, each with a value of (byte_position % 256)
  - This is part of the first HW assignment