Streams

Cpt S 321

Washington State University

Sample Scenario: Three Programmers

There are 3 different programmers in 3 different companies who all have similar jobs. They are all working on word processing applications and each one is responsible for writing the code that saves the document data to a file. After a short amount of time they each get the basic save code working, and move on to new tasks. But they also have responsibility for the maintenance of the document saving code.

Each of the 3 programmers writes roughly equivalent code that makes ~70 fwrite calls in order to save a file.

Requirements Change

- Then the requirements change and the programmers are told that in addition to saving to disk, the saving code must also support displaying the data in the console/terminal window.
- About a week after that requirement is introduced, each is told that they also need to support writing the data to a socket (which is a network communications mechanism, if you're not already familiar with sockets).
- Yet another week after that a fourth requirement is introduced, where the save code must also support writing the data to an encrypted file.
- Some time passes as the programmers work on their implementations…

How would you handle this?

Programmer Survey

Each programmer is interviewed afterwards to discuss how they dealt with their implementations and the frequent changes in requirements.

Programmer #1 (exhausted!)

“First I just had to write code to save the file to disk. There were only about 70 fwrite calls involved in this and it wasn’t too bad. Then they told me that they also wanted the option to display the save data to the terminal window so I had to go in and replace 70 instances of fwrite calls with if/then statements that made fwrite calls if we were writing to a file and printf calls if we weren’t. I also obviously had to introduce a new variable to keep track of which of the two methods we were using. Then when they said they needed support for saving to a socket I again had to modify 70 instances of fwrite/printf calls and add a third condition in. I had used a Boolean variable earlier which I had to change because now there were 3 possible save states instead of two. Also, I had to implement network I/O type code and that’s really not my background. Then when they introduced the fourth requirement I went through this process yet again.”

Programmer #2 (annoyed)

“In the beginning I had about 70 fwrite calls. When they introduced the requirement to display in the terminal window I did a find-and-replace to turn each fwrite call into a WriteData call. WriteData is the name of a function that I made to write data to the appropriate source. So then as each additional requirement came in, those being the socket and encryption requirements, I only had to modify code within a single function. It was still annoying that I had to write more code for each request though. The company has a specific file encryption scheme that I needed to learn about in order to properly implement the encrypted saving, which took a lot of my time. I just have this feeling like they’ll end up making even more requests for changes in the future and I’ll have to alter my code yet again.”

Programmer #3 (not at all mad!)

When interviewed, they stated that they wrote the saving code only once and never had to make any changes for the terminal, socket, or encryption requirements. When asked if this meant that all four options were implemented from the beginning, even before some of them were requested, Programmer 3 said no, and mentioned the following:

“I did not write a bunch of extra code to handle all those methods. I didn’t need to. I did it the smart way from the beginning and the other members of my team have been able to use my code to save to disk, write to the terminal window, write to a socket, and write to an encrypted file. But I didn’t write a single line of code that’s specific to any of those methods. The way I wrote it makes it so that I didn’t have to change a thing to support new saving methods.”

This is a big part of the course, i.e., about…

- Programmer 3 (P3) save themselves a lot of work
- Other members of his organization don’t have to change P3’s code in order to extend its ability to save to new sources
- Other members of the organization therefore don’t have to wait for P3 to make changes before they can move forward with their own work
- Time (and money) is saved throughout the organization and people’s jobs are easier
- But how was this achieved?

Programmer #3 Used Streams

```
bool Save(Stream s)
{
    …
    s.Write(data); // Use the write member function of the stream
    …
}
```

- Sounds familiar?

Recall the BasicMessageClass

- Our first solution:

```
public void ShowMessageConsole() => Console.WriteLine(message);
```

- Our improved solution:

```
public void ShowMessage(TextWriter textWriter) => textWriter.WriteLine(message);
```

What is a Stream?

- Abstract data type (abstract Stream class on MSDN)
- Conceptually represents a linear sequence of **bytes**
- Operations / 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 .NET, 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
  - Check the CanRead and CanWrite properties in Stream
- Other useful properties
  - CanSeek
  - Length
  - …

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 is 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

Back to the Scenario

- If you write a method to save data to a file and it takes a Stream as the parameter for the destination data, then the outside world can pass in a FileStream to write the data to a file, a CryptoStream to write to the data encrypted, or a NetworkStream to write (send) the data over a network.
  - All in .NET, so no programmers in the company spend extra time to get this functionality
- If it is desired to write the data to some other source, programmers can write classes that inherit from Stream and pass those in to the Save function.
  - Endless possibilities for where you can write the data
  - Again, no code is changed in the actual Save method written by the third programmer

TextReader and TextWriter Classes

- From the TextWriter documentation on MSDN: “Represents a writer that can write a sequential series of characters. This class is abstract.”
- Same idea as a stream in the sense that it’s an abstraction for a data source. In this case a sequence of text characters instead of bytes.
- TextReader
  - Abstraction for something that supports reading text
  - Has Read and ReadLine methods
  - Console.In is a TextReader
- TextWriter
  - Abstraction for something that supports writing text
  - Has Write and WriteLine methods
  - Console.Out is a TextWriter

What inherits from TextReader?

- System.IO.StreamReader class – has a constructor (one of many) that takes a string as a file name and can be used to read text from a file.

- System.IO.StringReader class – has only one constructor that takes a string as a parameter
  - Remember that TextReader has a ReadLine method, so if you have a string representing multiple lines of text from a text box in an interface, you can use this to read it line by line.

What Inherits from TextWriter?

- StreamWriter and StringWriter

- Analogous to StreamReader and StringReader

- StreamWriter can write lines to a text file:

```
using (StreamWriter outfile = new StreamWriter("somefile.txt"))
{
    outfile.WriteLine("Hello World!");
} // What does the “using” do?
```

Key Points

- If you have a scenario where you are loading from or saving to a file, use streams.
- You can have multiple overloads so that there is still a function that takes a string file name:

```
public void Save(string fileName)
{
   // NO saving here
   FileStream fs = new FileStream(
      fileName,
      FileMode.Create,
      FileAccess.Write);
   Save(fs);
   fs.Dispose();
}
```



```
public void Save(Stream s)
{
   // Implement the actual saving logic here
}
```

Time for coding!

- Create a **HelloWorldFileStream** project in the in-class exercise solution
- The main program should prompt the user for a file name
- Create a text file with that name
- Ask the user to provide the text that should be written in that file and do it
- If something goes wrong the user should be given the possibility to start over

- Remember: DESIGN FOR CHANGE!