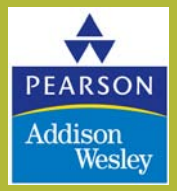
Chapter 10A:
File Streams and Exceptions

C# Software Solutions
Foundations of Program Design
First Edition

by
John Lewis



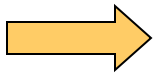
Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Files and Streams

- Some of the power of a program comes from its ability to remember information from one run to the next
- Chapter 10A focuses on:
  - Files and streams in text files
  - Streams for output
  - Streams for input
  - I/O Exceptions
  - Comma separated Files
  - Object Streams

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Outline



Files and Streams

Streams for output

Streams for input

I/O Exceptions

Comma separated Files

Object Streams

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

File Streams

- A *stream* is a sequence of bytes that flow from a source to a destination
- In a program, we read information from an input stream and write information to an output stream
- A program can manage multiple streams simultaneously

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Standard I/O

- There are three standard I/O streams:
  - *standard output* – defined by `Console.Out`
  - *standard input* – defined by `Console.In`
  - *standard error* – defined by `Console.Error`
- We use `Console.out` when we execute `WriteLine` statements
- `Console.out` and `Console.Error` typically represent a particular window on the monitor screen
- `Console.In` typically represents keyboard input, which we've used many times

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Writing Text Files

- In Chapter 5 we explored the use of the `StreamReader` class to read input from a text file
- Let's now examine other classes that let us write data to a text file
- The `StreamWriter` class represents a text output file, but with minimal support for manipulating data
- `System.IO` is an object of the `TextWriter` class, which provides `Write` and `WrintLine` methods defined for them

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Streams for Output

- If we want our program to write text to a disk file, it must create a new file or it must write to an existing file

- `using System.IO;`
- …
- `string outFileName = “myData.txt”;`
- `StreamWriter outStream = new StreamWriter(outFileName);`
- Or
- `StreamWriter outStream = File.CreateText(outFileName);`

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Streams for Output

- Now you can use `WriteLine` or `Wrint` to write a line to the file

- `outStream.WriteLine(“this text will…”);`
- `outStream.Write(“This text will…\r\n”);`

- And Finally close the file
- `outStream.Close();`

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Streams for Output

- If the file exists, it can be either overwritten or appended to

- `string outFileName = “myData.txt”;`

- ```
  StreamWriter outStream = new
                           StreamWriter(outFileName);
  ```

- ```
  StreamWriter outStream = new
                       StreamWriter(outFileName, true);
  ```

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Writing Text Files

- Finally, we'll also use the `StreamWriter` class for advanced internationalization and error checking
- We build the class that represents the output file by combining these classes appropriately
- See `TestData.cs`

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

TestData.cs

Listing 10.7

```
//********************************************************************
//  TestData.cs        C#:  Ken Culp
//
//  Demonstrates the use of a character file output stream.
//********************************************************************
using System;
using System.IO;

namespace TestData
{
  public class TestData
  {
    //-----------------------------------------------------------
    //  Creates file of test data that consists of ten lines each
    //  containing ten integer values in the range 10 to 99.
    //-----------------------------------------------------------
    public static void Main(string[] args)
    {
      const int MAX = 10;
      int value;

      // Relative path places the file in the Solution (project)
      // directory (up two directories from bin\Debug).
      string file = @"..\..\test.dat";
```

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

TestData.cs

Listing 10.7 continued

```
      Random rand = new Random(DateTime.Now.Millisecond);
      StreamWriter outFile = new StreamWriter(file);

      for (int line = 1; line <= MAX; line++)
      {
        for (int num = 1; num <= MAX; num++)
        {
          value = rand.Next(90) + 10;
          outFile.Write(value + "   ");
        }
        outFile.WriteLine();
      }
      outFile.Close();
      Console.Out.WriteLine("Output file has been created: " +
        file);

      Console.In.ReadLine();  // Wait for enter key
    }
  }
}
```

Output

```
Output file has been created: ..\..\test.dat
```

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Streams for Input

- If a text file already exists and our progra needs to read from it, we should create an instance of `StreamReader`

- `using System.IO;`
- …
- `string inFileName = “myData.txt”;`
- `StreamReader inStream = new StreamReader(inFileName);`
- Or
- `StreamReader inStream = File.OpenText(inFileName);`

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Streams for Input

- Now you can use `ReadLine` or `Read` to read a line or a character from the file

- `string s = inStream.ReadLine();`

- `int k = 0;`
- `while((k = inStream.Read()) != -1){`
- `    char c = (char)k;`
- `    k++;`
- `}`
- And Finally close the file
- `inStream.Close();`

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

The IOException Class

- Operations performed by some I/O classes may throw an `IOException`
  - A file might not exist
  - Even if the file exists, a program may not be able to find it
  - The file might not contain the kind of data we expect
- `IOException` is defined in the `System.IO` namespace

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

File Exception

- Several Exception types exists, which would be applicable when opening a file
- `IOException`
- `DirectoryNotFoundException`
- `FileLoadException`
- `FileNotFoundException`
- `PathTooLongException`
- And many more

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Comma separated Files

- Often so called *comma separated* files are used
- In such files strings are separated by comma or another character
- Comma separated files can be used as a simple form of a data base
- Spreadsheets can be converted to comma separated files and read into your program
- When read into your program, separated strings should be placed in a String Array

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Comma separated Files

- Example:

```
string[] words = new string[10];
char[] separator = {‘,’};
string line = inStream.ReadLine();
while(line != null){
    words = line.Split(separator);
    Console.Out.WriteLine(words[0].Trim());
    …
}
```

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Object Streams

- By use of *Object Streams* you can save and store objects on files
- Objects has to be *Serialized* when saved and *Deserialized* when loaded
- The attribute `[Serializable]` must be placed in front of the class definition for objects which should be saved on files.

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Object Streams example

```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

namespace ObjectStreamTest
{
    public class ObjectStream
    {
        static public void Save(object data)
        {
            FileStream file = new FileStream("store.bin", FileMode.Create);
            BinaryFormatter formatter = new BinaryFormatter();
            formatter.Serialize(file, data);
            file.Close();
        }

        static public object Load()
        {
            FileStream file = new FileStream("store.bin", FileMode.OpenOrCreate);
            object data = null;
            if (file.Length != 0)
            {
                BinaryFormatter formatter = new BinaryFormatter();
                data = formatter.Deserialize(file);
            }
            file.Close();
            return data;
        }
    }
}
```

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Summary

- Chapter 10A has focused on:
  - Files and streams in text files
  - Streams for output
  - Streams for input
  - I/O Exceptions
  - Comma separated Files
  - Object Streams

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley