C# Data Manipulation

Hans-Wolfgang Loidl
<H.W.Loidl@hw.ac.uk>

**School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh**

Semester 1 — 2018/19

HERIOT WATT UNIVERSITY

The Stream Programming Model

- File streams can be used to access stored data.
- A stream is an object that represents a generic sequence of bytes.
- Any type of data, marked Serializable, can be transformed into a stream. This is called *serialisation*
- Streams can then be used to:
  - Read/Write data from/to disk.
  - Move data between machines.
- Although streams work at the byte level, programmers don't need to work with bytes.
- Reader and Writer objects are usually used to ease the use of streams.

HERIOT WATT UNIVERSITY

Manual serialisation

- Writing your own serialisation function is easy, and useful in many different contexts, eg. implementing `ToString()`.
- To serialise an object of class A:
  - Serialise all value type attributes, by directly writing the data into the result buffer
  - Serialise all reference types attributes by recursively calling serialisation on them.

HERIOT WATT UNIVERSITY

Naive serialisation

We implement `ToString()` for our Person/Student example as one special case of serialisation:

```
1 public string ToString0() {
2   return String.Format(
3       "Name: {0} {1}\tAddress: {2}\nMatricNo: {3}\
        tDegree: {4}",
4                                this.GetfName(),
5           this.GetlName(),
6           this.GetAddress(),
7           this.matricNo,
8           this.degree);
9 }
```

What's the disadvantage with this implementation?

HERIOT WATT UNIVERSITY

Naive serialisation

We implement `ToString()` for our Person/Student example as one special case of serialisation:

```
1 public string ToString0() {
2   return String.Format(
3       "Name: {0} {1}\tAddress: {2}\nMatricNo: {3}\
        tDegree: {4}",
4                                this.GetfName(),
5           this.GetlName(),
6           this.GetAddress(),
7           this.matricNo,
8           this.degree);
9 }
```

What's the disadvantage with this implementation?

An example of serialisation

This is a better implementation of serialisation:

```
1 public override string ToString() {
2  string base_str = base.ToString();
3  string this_str = String.Format(
4     "MatricNo: {0}\tDegree: {1}",
5      this.matricNo, this.degree);
6  return base_str+"\n"+this_str;
7 }
```

HERIOT WATT UNIVERSITY

Accessing files using streams

- Generate a Reader/Writer object
- This internal generates a stream object
- This object directly interacts with the file
- Closing the Reader/Writer object, also closes the internal stream object

HERIOT WATT UNIVERSITY

C# Support for File Streams

- C# provides a number of *abstract* classes in the `System.IO` namespace to access data in files including `Stream, TextWriter and TextReader`.
- The stream class is used to access data at the byte level.
- `TextWriter` and `TextReader` support access to readable text through using
  - `Write()` and `WriteLine()` of `TextWriter`.
  - `Read()` and `ReadLine()` of `TextReader`.
- Several classes derive from these abstract classes, and implement customised versions of reading and writing:
  - `StreamReader` and `StreamWriter` for text data
  - `BinaryReader` and `BinaryWriter` for binary data

HERIOT WATT UNIVERSITY

Example: Accessing a File

```
1  using System;
2  using System.IO;
3
4  public class FileReadWrite{
5    public static void Main(){
6      // Write to a file
7      StreamWriter sw = new StreamWriter("test.txt");
8      sw.Write("Hello World!");
9      sw.Close();
10
11     // Reading from a file
12     StreamReader sr = new StreamReader("test.txt");
13     Console.WriteLine(sr.ReadLine());
14     sr.Close();
15   }
16 }
```

HERIOT WATT UNIVERSITY

More on File Access

Reading from a file line-by-line:

```
1 StreamReader sr = new StreamReader("test.txt");
2 string inValue = "";
3 while((inValue = sr.ReadLine()) != null)
4   Console.WriteLine(inValue);
```

HERIOT WATT UNIVERSITY

Handling file access problems with exceptions

Body that is executed

```
1 try {
2   StreamWriter sw = new StreamWriter("test.txt");
3   sw.Write("Hello␣World!");
4   sw.Close();
5 } catch(IOException ex) {
6   Console.WriteLine(ex.Message);
7 }
```

Catch block, executed if `IOException` was raised

HERIOT WATT UNIVERSITY

Handling file access problems with exceptions

Body that is executed

```
1 try {
2   StreamWriter sw = new StreamWriter("test.txt");
3   sw.Write("Hello␣World!");
4   sw.Close();
5 } catch(IOException ex) {
6   Console.WriteLine(ex.Message);
7 }
```

Catch block, executed if `IOException` was raised

HERIOT WATT UNIVERSITY

Handling file access problems with exceptions

Body that is executed

```
1 try {
2   StreamWriter sw = new StreamWriter("test.txt");
3   sw.Write("Hello␣World!");
4   sw.Close();
5 } catch(IOException ex) {
6   Console.WriteLine(ex.Message);
7 }
```

Catch block, executed if `IOException` was raised

HERIOT WATT UNIVERSITY

Another common pattern

```
using (StreamReader sr = new StreamReader(infile)) {
    // open file
  using (StreamWriter sw = new StreamWriter(outfile))
      {
    string str = "";
    string str0  = "";
    while ((str = sr.ReadLine()) != null)// iterate
        over lines
    {
      str0 = "";
      foreach (char c in str) {
        if (Char.IsPunctuation(c)) {
          // nothing
        } else {
          str0 += c;
        }
      }
      sw.WriteLine(str0.ToLower());
    }
```

Read line-by-line

Remove punctuation

Write to different file

Another common pattern

```
using (StreamReader sr = new StreamReader(infile)) {
    // open file
  using (StreamWriter sw = new StreamWriter(outfile))
      {
    string str = "";
    string str0  = "";
    while ((str = sr.ReadLine()) != null)// iterate
        over lines
    {
      str0 = "";
      foreach (char c in str) {
        if (Char.IsPunctuation(c)) {
          // nothing
        } else {
          str0 += c;
        }
      }
      sw.WriteLine(str0.ToLower());
    }
```

Read line-by-line

Remove punctuation

Write to different file

Another common pattern

```
1  using (StreamReader sr = new StreamReader(infile)) {
       // open file
2    using (StreamWriter sw = new StreamWriter(outfile))
         {
3      string str = "";
4      string str0  = "";
5      while ((str = sr.ReadLine()) != null)// iterate
           over lines
6      {
7        str0 = "";
8        foreach (char c in str) {
9          if (Char.IsPunctuation(c)) {
10           // nothing
11         } else {
12           str0 += c;
13         }
14       }
15       sw.WriteLine(str0.ToLower());
16     }
```

Read line-by-line

Remove punctuation

Write to different file

Another common pattern

```
using (StreamReader sr = new StreamReader(infile)) {
    // open file
  using (StreamWriter sw = new StreamWriter(outfile))
      {
    string str = "";
    string str0  = "";
    while ((str = sr.ReadLine()) != null)// iterate
        over lines
    {
      str0 = "";
      foreach (char c in str) {
        if (Char.IsPunctuation(c)) {
          // nothing
        } else {
          str0 += c;
        }
      }
      sw.WriteLine(str0.ToLower());
    }
```

Read line-by-line

Remove punctuation

Write to different file

Summary

- Stream programming in general deals with serialising and transfering data
- One example is reading/writing from/to files
- Other examples are transfering data over a network or a persistent storage
- The basic interface for file access is provide by hte `System.IO` namespace through `StreamReader and StreamWriter`

HERIOT WATT UNIVERSITY