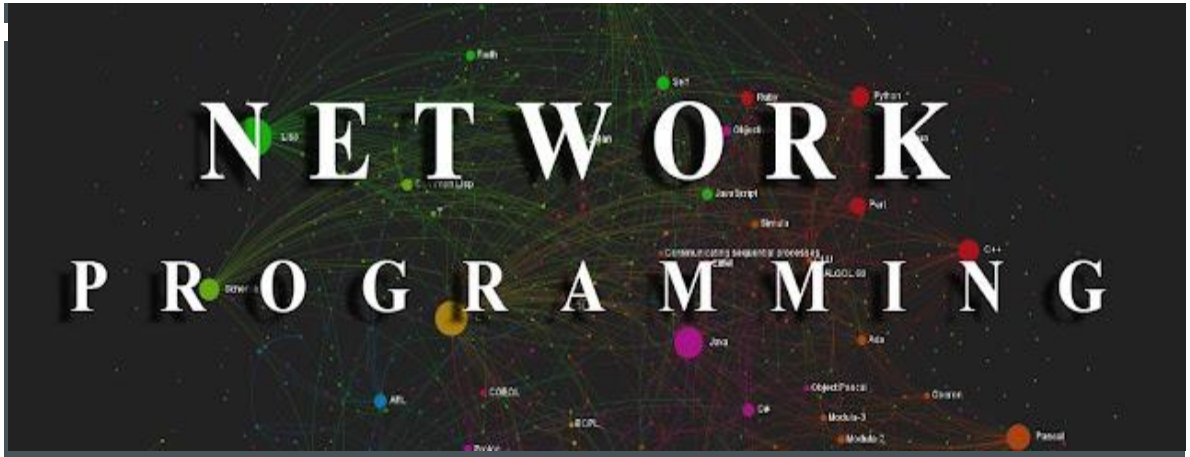


IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

ALI.HUSSEIN@AUN.EDU.EG

IT DEPT - OFFICE NO. 312

2022

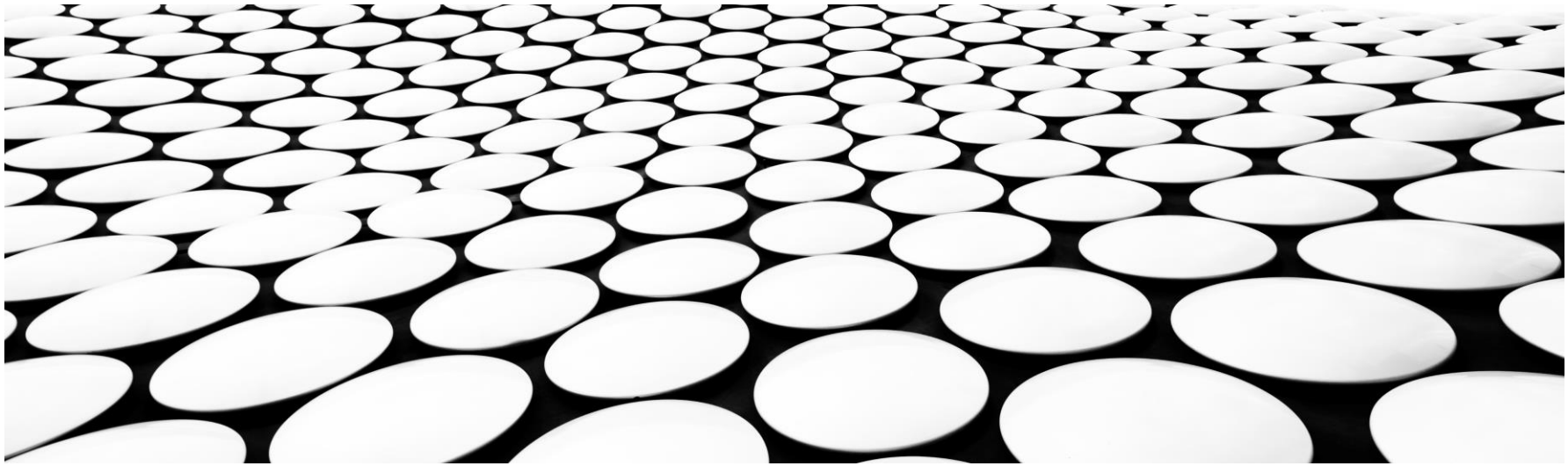
جامعة أسيوط

كلية الحاسبات والمعلومات

كلية معتمدة من الهيئة القومية لضمان الجودة والاعتماد

LECTURE 1

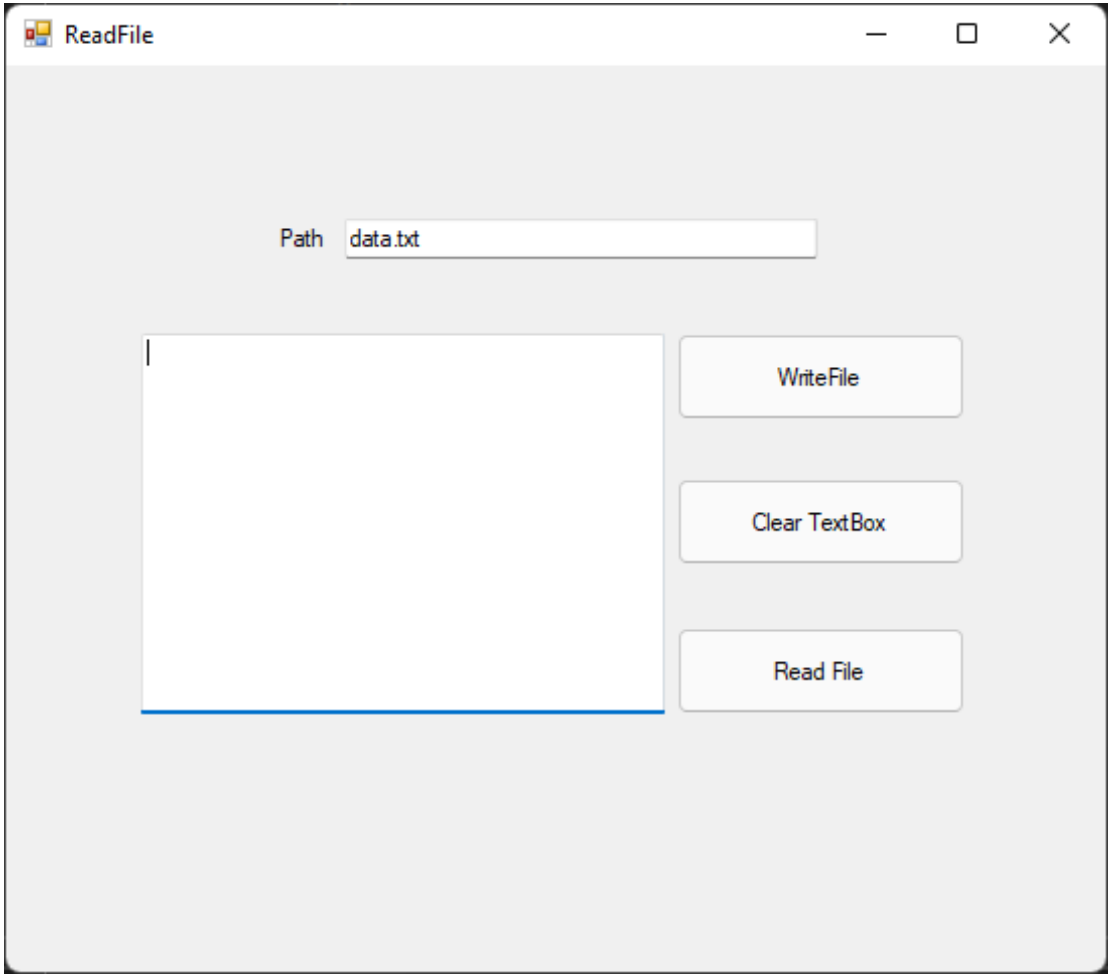
PROGRAMMING THE STREAMS



ITS TIME TO CODE !

- Be ready for some examples

EXAMPLE: FILESTREAMS (READ AND WRITE)



Our GUI

WRITE 3 CHARS

```
private void button2_Click(object sender, EventArgs e)
    {
        Stream s = new FileStream(txtPath.Text, FileMode.Create);
        s.WriteByte(79);  // O    01001111
        s.WriteByte(79);  // O    01001111
        s.WriteByte(80);  // P    01010000
        s.Close();
    }
```

READ MORE THAN 3 CHARS

```
private void button1_Click(object sender, EventArgs e)
    {
        Stream s = new FileStream(txtPath.Text, FileMode.Open);
        int i, j, k, m, n;
        i = s.ReadByte();  // O   79  01001111
        j = s.ReadByte();  // O   79  01001111
        k = s.ReadByte();  // P   80  01010000

        m = s.ReadByte();  // -1  EOF
        n = s.ReadByte();  // -1  EOF

        txtData.Text = String.Format("{0} {1} {2} {3} {4}", i, j, k, m, n);
        s.Close();
    }
```

LAB

- Extend the code to read and write any number of chars, make the same previous GUI functional (problem 1 sheet2)

NOTE

Class **FileStream** offers binary input and output

PRACTICE HINT

- The **using** control structure is helpful when we do IO programming
- Syntax : **using** (*type variable = initializer*) *body*
- Semantics
  - In the scope of using, bind variable to the value of initializer
  - The type must implement the interface **IDisposable**
  - Execute body with the established name binding
  - At the end of body do variable Dispose automatically
    - Usually the Dispose methods in the subclasses of Stream call Close
    - ***Why we need this ?***

SYNTAX

```
using (Stream s = new FileStream("myFile.txt", FileMode.Create))
        {
            s.WriteByte(79);  // O   01001111
            s.WriteByte(79);  // O   01001111
            s.WriteByte(80);  // P   01010000
        }
```

WHY USING

- // The using statement …

- using (type variable = initializer)
- body

- {type variable = initializer;
- try {
- body
- }
- finally {
- if (variable != null)
- ((IDisposable)variable).Dispose();
- }
- }

CONSOLE APPLICATION TO COPY FILES

- This example copies the content of a file to another.
- A console application: files names are passed as a command line arguments.
  - `copy-file source-file.txt target-file.txt`
- During your lab, convert it to WinForm APP, problem 2 sheet 2
- Make your GUI Attractive, Plz

```
using System;
using System.IO;
public class CopyApp
{
    public static void Main(string[] args) {
        FileCopy(args[0], args[1]);
    }
    public static void FileCopy(string fromFile, string toFile) {
        try {
            using (FileStream fromStream =  new FileStream(fromFile, FileMode.Open))
            {
                using (FileStream toStream = new FileStream(toFile, FileMode.Create))
                {
                    int c;
                    do       {
                        c = fromStream.ReadByte();
                        if (c != -1) toStream.WriteByte((byte)c);
                    } while (c != -1);
                }   }   }
        catch (FileNotFoundException e)
        {
            Console.WriteLine("File {0} not found: ", e.FileName);
            throw;
        }
        catch (Exception)
        {
            Console.WriteLine("Other file copy exception");
            throw;
        }  }  }
```

ADAPTER CLASSES

- A set of classes provided by FCL to support reading and writing in higher level

	Input	Output
Text	***TextReader (base)*** **StreamReader** **StringReader**	***TextWriter(base)*** **StreamWriter** **StringWriter**
Binary	**BinaryReader**	**BinaryWriter**

ADAPTER CLASSES

- Higher level of abstraction
  - IO of chars and text strings - not just raw bytes
  - IO of values in simple types
- The *TypeReader* and *TypeWriter* classes are not subclasses of the stream classes
  - They are typically built on - and delegates to - a Stream object
  - A Reader or Writer classes has a stream - it is not a stream
  - Reader and Writer classes serve as Stream adapters

THE CLASS ENCODING

- An encoding is a mapping between characters/strings and byte arrays
- An object of class ***System.Text.Encoding*** represents knowledge about a particular character encoding

THE CLASS ENCODING

- byte[] GetBytes(string) **Instance method**
- byte[] GetBytes(char[]) **Instance method**
  - Encodes a string/char array to a byte array relative to the current encoding
- char[] GetChars(byte[]) Instance method
  - Decodes a byte array to a char array relative to the current encoding
- byte[] Convert(Encoding, Encoding, byte[]) Static method
  - Converts a byte array from one encoding (first parameter) to another encoding (second parameter)

IT TIME TO CODE!

- The following example shows how you can use the Encoding class for either converting to and from byte and also convert between different encodings.
- From a unicode string to a byte array in a given encoding ***(GetBytes - Encoding).***
- From a byte array in one encoding to a byte array in another encoding (Convert)
- From a byte array in a given encoding to a char array (in unicode)

  (GetChars - Decoding).
- From a char array (in unicode) to a unicode string encoding.

  (via String constructor)

```
using System;
using System.Text;
/* Adapted from an example provided by Microsoft */
class ConvertExampleClass{
   public static void Main()    {
      string unicodeStr =          // "A æ u å æ ø i æ å"
         "A \u00E6 u \u00E5 \u00E6 \u00F8 i \u00E6 \u00E5";
      // Different encodings.
      Encoding ascii = Encoding.ASCII,  unicode = Encoding.Unicode,
               utf8 = Encoding.UTF8,
               isoLatin1 = Encoding.GetEncoding("iso-8859-1");

      // Encodes the characters in a string to a byte array:
      byte[] unicodeBytes = unicode.GetBytes(unicodeStr),
            asciiBytes = ascii.GetBytes(unicodeStr),
            utf8Bytes = utf8.GetBytes(unicodeStr),
            isoLatin1Bytes = utf8.GetBytes(unicodeStr);

      // Convert from byte array in unicode to byte array in utf8:
      byte[] utf8BytesFromUnicode =
        Encoding.Convert(unicode, utf8, unicodeBytes);
```

```
    // Convert from byte array in utf8 to byte array in ascii:
    byte[] asciiBytesFromUtf8 =
      Encoding.Convert(utf8, ascii, utf8Bytes);

    // Decodes the bytes in byte arrays to a char array:
    char[] utf8Chars = utf8.GetChars(utf8BytesFromUnicode);
    char[] asciiChars = ascii.GetChars(asciiBytesFromUtf8);

    // Convert char[] to string:
    string utf8String = new string(utf8Chars),
           asciiString = new String(asciiChars);

    // Display the strings created before and after the conversion.
    Console.WriteLine("Original string: {0}", unicodeStr);
    Console.WriteLine("String via UTF-8: {0}", utf8String);
    Console.WriteLine("ASCII converted string: {0}", asciiString);
  }
}
```

Microsoft Visual Studio Debug Console

```
Original string: A æ u å æ o i æ å
String via UTF-8: A æ u å æ o i æ å
ASCII converted string: A ? u ? ? ? i ? ?

C:\Users\Dr. Ali\source\repos\WindowsFormsApp1\Encod
.
Press any key to close this window . . .
```

OUTPUT, JUSTIFY THE “?”

THE CLASS TEXTWRITER

- Class TextWriter supports writing of characters and strings via a chosen encoding
- It is also possible to write textual representations of simple types with use of TextWriter.
- Subclasses of the abstract TextWriter class
  - StreamWriter: For character output in a particular character encoding
  - StringWriter: For stream access to a string
- Plz code the related problem in Sheet 2

MEMBERS IN CLASS STREAMWRITER

- 7 overloaded constructors
  - Parameters involved: File name, stream, encoding, buffer size
  - `StreamWriter(String)`
  - `StreamWriter(Stream)`
  - `StreamWriter(Stream, Encoding)`
  - *others*
- 17/18 overloaded Write / WriteLine operations
  - Chars, strings, simple types. Formatted output
- `Encoding`
  - A property that gets the encoding used for this `TextWriter`
- `NewLine`
  - A property that gets/sets the applied newline string of this `TextWriter`
- *others*

ADAPTER CLASSES CONT.

- Class TextReader supports reading of characters via a chosen encoding
- Class TextReader does not have methods to read textual representation of simple types
- Subclasses of the abstract TextReader class
  - StreamReader: For character input in a particular character encoding
  - StringReader: For stream access to a string - (discussed later in the lecture)

PROGRAM

- Reading back the text strings encoded in three different ways, with StreamReader.
- In the last half part of the program, the binary contents of the three files are read and reported.

SELF CODE STUDY

- Plz download the cs file from my drive and run it
- plz download and run me !

StreamReader MEMBERS

- 10 StreamReader constructors
  - Similar to the StreamWriter constructors
  - **StreamReader(String)**
  - **StreamReader(Stream)**
  - **StreamReader(Stream, bool)**
  - **StreamReader(Stream, Encoding)**
  - *others*
- **int Read()** Reads a single character. Returns -1 if at end of file
- **int Read(char[], int, int)** Returns the number of characters read
- **int Peek()**
- **String ReadLine()**
- **String ReadToEnd()**
- **CurrentEncoding**
  - A property that gets the encoding of this StreamReader

THE BINARYWRITER, AND BINARYREADER CLASSES

- Class BinaryWriter allows us to write values of simple types in binary format
- Encodings are only relevant if values of type char are written
- Class BinaryReader allows us to read values of simple types in binary format
- Symmetric to BinaryWriter

BinaryWriter MAIN METHODS AND CONSTRUCTORS

- Two public constructors
  - BinaryWriter(Stream)
  - BinaryWriter(Stream, Encoding)
- 18 overloaded Write operations
  - One for each simple type
  - Write(char),
  - Write(char[]), and Write(char[], int, int) - use Encoding
  - Write(string) - use Encoding
  - Write(byte[]) and Write(byte[], int, int)
- Seek(int offset, SeekOrigin origin)

BinaryReader **MAIN METHODS AND CONSTRUCTORS**

- Two public constructors
  - BinaryReader(Stream)
  - BinaryReader(Stream, Encoding)
- 15 individually name Read*type* operations
  - ReadBoolean, ReadChar, ReadByte, ReadDouble, ReadDecimal, ReadInt16, ...
- Three overloaded Read operations
  - Read() and Read (char[] buffer, int index, int count)
    read characters - using Encoding
  - Read (bytes[] buffer, int index, int count) reads bytes

EXAMPLES

EXAMPLE 1 , BinaryWriter

```
using System;
using System.IO;

public class BinaryWriteSimpleTypes{

  public static void Main(){
    string fn = "simple-types.bin";

    using(BinaryWriter bw =
          new BinaryWriter( new FileStream(fn, FileMode.Create))){
      bw.Write(5);      // 4  bytes
      bw.Write(5.5);    // 8  bytes
      bw.Write(5555M);  // 16 bytes (decimal dt)
      bw.Write(5==6);   // 1  bytes
    }

    FileInfo fi = new FileInfo(fn);
    Console.WriteLine("Length of {0}: {1}", fn, fi.Length);
  }
}
```

```
Length of simple-types.bin: 29
```

EXAMPLE 2 , BinaryReader

```
using System;
using System.IO;

public class BinaryReadSimpleTypes{

  public static void Main(){
    string fn = "simple-types.bin";

    using(BinaryReader br =
          new BinaryReader(    new FileStream(fn, FileMode.Open))){
      int i = br.ReadInt32();
      double d = br.ReadDouble();
      decimal dm = br.ReadDecimal();
      bool b = br.ReadBoolean();

      Console.WriteLine("Integer i: {0}", i);
      Console.WriteLine("Double d: {0}", d);
      Console.WriteLine("Decimal dm: {0}", dm);
      Console.WriteLine("Boolean b: {0}", b);
    } } }
```

```
Integer i: 5
Double d: 5,5
Decimal dm: 5555
Boolean b: False
```

The **File** and **FileInfo** classes

- The class **FileInfo** represents a file
- The class **File** holds static methods for creation, copying, deletion, moving, and opening of files

Example: the **FileInfo** classes

```
using System;
using System.IO;
public class FileInfoDemo{
  public static void Main(){
    // Setting up file names
    string fileName = "file-info.cs",  fileNameCopy = "file-info-copy.cs";

    // Testing file existence
    FileInfo fi = new FileInfo(fileName); // this source file
    Console.WriteLine("{0} does {1} exist",  fileName, fi.Exists ? "" : "not");

    // Show file info properties:
    Console.WriteLine("DirectoryName: {0}", fi.DirectoryName);
    Console.WriteLine("FullName: {0}", fi.FullName);
    Console.WriteLine("Extension: {0}", fi.Extension);
    Console.WriteLine("Name: {0}", fi.Name);
    Console.WriteLine("Length: {0}", fi.Length);
    Console.WriteLine("CreationTime: {0}", fi.CreationTime);

    // Copy one file to another
    fi.CopyTo(fileNameCopy);
    FileInfo fiCopy  = new FileInfo(fileNameCopy);
```

Example (Cont): The **File** and **FileInfo** classes

```
// Does the copy exist?
    Console.WriteLine("{0} does {1} exist", fileNameCopy, fiCopy.Exists ? "" : "not");

    // Delete the copy again
    fiCopy.Delete();

    // Does the copy exist?
    Console.WriteLine("{0} does {1} exist",
                    fileNameCopy, fiCopy.Exists ? "" : "not"); // !!??

    // Create new FileInfo object for the copy
    FileInfo fiCopy1  = new FileInfo(fileNameCopy);
    // Check if the copy exists?
    Console.WriteLine("{0} does {1} exist", fileNameCopy, fiCopy1.Exists ? "" : "not");

    // Achieve a TextReader (StreamReader) from the file info object
    // and echo the first ten lines in the file to standard output
    using(StreamReader sr = fi.OpenText()){
      for (int i = 1; i <= 10; i++)
        Console.WriteLine("  " + sr.ReadLine());
    }   }  }
```

Program: A demonstration of the File class.

```
using System;
using System.IO;
public class FileDemo
{
   public static void Main()
   {
      // Setup file names
      string fileName = "binarystreams.exe", // this source file
      fileNameCopy = "fileCopy.cs";

      // Does this source file exist?
      Console.WriteLine("{0} does {1} exist",
      fileName, File.Exists(fileName) ? "" : "not");

      // Copy this source file
      File.Copy(fileName, fileNameCopy);

      // Does the copy exist?
      Console.WriteLine("{0} does {1} exist", fileNameCopy,
      File.Exists(fileNameCopy) ? "" : "not");

      // Delete the copy again
      Console.WriteLine("Deleting {0}", fileNameCopy);
      File.Delete(fileNameCopy);

      // Does the deleted file exist
      Console.WriteLine("{0} does {1} exist", fileNameCopy,
      File.Exists(fileNameCopy) ? "" : "not");

      // Read all lines in source file and echo
      // one of them to the console
      string[] lines = File.ReadAllLines(fileName);
      Console.WriteLine("Line {0}: {1}", 6, lines[6]);    }}
```

- The class **DirectoryInfo** represents a directory
- The class **Directory** holds static methods for creating, moving, and enumerating through directories

Program: A demonstration of the DirectoryInfo class.

Do you note anything ?

```
using System;
using System.IO;
public class DirectoryInfoDemo
{
   public static void Main()
   {
      string fileName = "directory-info.cs";    // The current source file

      // Get the DirectoryInfo of the current directory
      // from the FileInfo of the current source file
      FileInfo fi = new FileInfo(fileName);     // This source file
      DirectoryInfo di = fi.Directory;

      Console.WriteLine("File {0} is in directory \n   {1}", fi, di);

      // Get the files and directories in the parent directory.
      FileInfo[] files = di.Parent.GetFiles();
      DirectoryInfo[] dirs = di.Parent.GetDirectories();

      // Show the name of files and directories on the console
      Console.WriteLine("\nListing directory {0}:", di.Parent.Name);
      foreach (DirectoryInfo d in dirs)
         Console.WriteLine(d.Name);
      foreach (FileInfo f in files)
         Console.WriteLine(f.Name);
   }
}
```

Program: A demonstration of the Directory class - similar to the DirectoryInfo demo program.

```
using System;
using System.IO;
public class DirectoryDemo
{
   public static void Main()
   {
      string fileName = "binarystreams.exe";      // The current source file
      FileInfo fi = new FileInfo(fileName);   // This source file

      string thisFile = fi.FullName,
            thisDir = Directory.GetParent(thisFile).FullName,
            parentDir = Directory.GetParent(thisDir).FullName;

      Console.WriteLine("This file: {0}", thisFile);
      Console.WriteLine("This Directory: {0}", thisDir);
      Console.WriteLine("Parent directory: {0}", parentDir);

      string[] files = Directory.GetFiles(parentDir);
      string[] dirs = Directory.GetDirectories(parentDir);

      Console.WriteLine("\nListing directory {0}:", parentDir);
      foreach (string d in dirs)
         Console.WriteLine(d);
      foreach (string f in files)
         Console.WriteLine(f);    }  }
```

Output in the next Slide

Output

```
This file: C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug\net6.0\binarystreams.exe
This Directory: C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug\net6.0
Parent directory: C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug

Listing directory C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug:
C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug\net6.0
```