Exercise 1, Streaming

Write a program that reads a text file and displays the number of words in the file.

Write a program that reads a text file and displays the longest word in the file.

Exercise 2, Serialization

Given the start of the class implementation below

```
class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Ssn { get; set; }
    ...
```

Finish the class implementation and make it serializable in order to be able to save and load the data.

Make a test program that uses a `BinaryFormatter` to save and load the data from a `Person` object to and from a file (remember to investigate the file).

Enhance your class `Person` so that it can be serialized as JSON. Expand on your test program to test this functionality (remember to investigate the file).

Hint: Have a look at `System.Runtime.Serialization` *and* `System.Text.Json`*. Alternatively, you can choose the third party library* `Newtonsoft.Json` *which is added as a NuGet package to your project.*

Optional: Instead of JSON, serialize your class `Person` using XML.

Hint: implement `IXmlSerializable` *and have a look at* `System.Xml` *and* `System.Xml.Serialization`

Exercise 3, Threading

Given the code below, which is a simple file reader class

```
using System;
using System.IO;

class Reader
{
    string fileName;
    public string Data { get; set; }

    public Reader(string fn) { fileName = fn; }

    public void Read()
    {
        FileStream s = new FileStream(fileName, FileMode.Open);
        StreamReader r = new StreamReader(s);
        Data = r.ReadToEnd();
        r.Close();
        s.Close();
    }
}
```

Write a C# program which uses the class above to compare two files. The files must be read in separate threads running in parallel.

The comparison should be based upon actual data in the files and write whether the files are identical in terms of data or different.

Exercise 4, Networking

a) In this exercise you must implement a TCP Server in a console application. The server should listen for incoming client connections. When a client connects to the server you must create a thread running the communication between the server and the client.

The following code can be used as inspiration. It creates a listener for localhost (127.0.0.1) and waits for a client to connect. When client connects it creates a network stream (retrieved from `TcpClient` class). This network stream is used to communicate between client and server.

```
byte[] adr = { 127, 0, 0, 1 };
IPAddress ipAdr = new IPAddress(adr);
TcpListener newsock = new TcpListener( ipAdr, 5000 );
newsock.Start();
Console.WriteLine("Waiting for a client...");

TcpClient client = newsock.AcceptTcpClient();
NetworkStream ns = client.GetStream();

string welcome = "Welcome to the DNP test server";
data = Encoding.ASCII.GetBytes(welcome);
ns.Write(data, 0, data.Length);
```

b) In a different folder, create a console application running as a TCP Client. The client should connect to the server and communicate, i.e. send data input in the console to the server.

Please notice: You should not name your project or class TcpClient, since it is the name of the class in the framework!

c) Work on the server from a) and implement a way to display messages from the connected client(s). Make an auto response to the client when data is received.