COMPUTING SUBJECT:	**Socket programming**
TYPE:	Assignment
IDENTIFICATION:	SocketHttpStart
COPYRIGHT:	*Michael Claudius* *Revised by Jamshid Eftekhari&Michael Claudius*
LEVEL:	Intermediate
TIME CONSUMPTION:	1-3 hours
EXTENT:	50 lines
OBJECTIVE:	TCP-sockets concurrent style Http command GET
PRECONDITIONS:	Computer Networks Ch. 2.7, 2.2
COMMANDS:	

IDENTIFICATION: SocketHttpStart

The Mission
We are going to explore how to interpret Http-command using a concurrent server.

Precondition
You have done the assignment SocketConcurrent and have a running concurrent solution

Useful C# links

- http://blogs.msdn.com/b/pfxteam/archive/2010/06/13/10024153.aspx
- https://msdn.microsoft.com/en-us/library/ms228388.aspx
- System.IO.StreamReader
- System.Net.Sockets.NetworkStream
- The method String.split(...), the first and simplest example is most important.
-

Now we will extend the server program to interpret the messages from the client.

Assignment 0. Project: HttpStart
Create a new project HttpStart and copy and paste the classes from SocketConcurrent into this project.

Assignment 1. Model class: ServiceEcho
If you have not done it yet, then extend the doIt method to send back "Server stopped" and then break the loop if it receives the message "STOP".
If you are lacking time then skip this for now.

Assignment 2 Model class: ServiceEcho
The server must now be able to understand more complicated messages; i.e. if a message is a Http-request this shall be recognized. Extend the *doIt()* method with the following responsibility:

> If the received message (request-line), looks like "GET /somefile.html HTTP/1.1" then extract the URI (middle) part of the request line (/somefile.html in this case) and include the URI in the response to the client.

Use your old TcpEchoClient program to test it.

Tip: Utilize the String.split() method to split the request-line.

Assignment 3 Model class: ServiceEcho
Still in the *doIt()* method change the response of a GET-request to look exactly like the first lines of Http-response without any data:

```
HTTP/1.1 200 OK\r\n
Content-Type: text/html\r\n"
Connection: close\r\n"
```

Finally add a line with some data like “Hello client”.

As several lines are now written to the client, the client-program must be changed to read an unknown number of messages from the server.

Assignment 4 Browser: Browser as a client
Start the server and then try to see the response in a browser.
To let the browser act as a client you use the url:

```
http://localhost:6789/somefile.html
```

No luck? Think carefully about the response format and \r\n.
Furthermore, how does the browser know that there is no more from the server?

Assignment 5: File handling
Make a text file somefile.html with some text lines like:

```
Hello Client
Hello World
```

Extend the *doIt()* method with the following responsibility:

> If the received message (request-line), looks like “GET /somefile.html HTTP/1.1”
> Then first send the header then open the file, read the content (line by line) and print out and send each line to the client.

Test it with the client and the browser.
IF you have problems with this assignment look at the Appendix A, next page.

One issue is that this program can only handle text-files.

Assignment 6: File handling
Change the DoIt method to utilize

- The method Stream.CopyTo(...)

Which is a byte stream and thus pictures can also be handled.

Appendix A

In assignment 5 you are asked that the server should send the contents of the resource (read "file") from the requests URI.
In the previous assignments of the server you extracted the URI from the request line. Now you must define where on the servers disk to look for this file. The place to start looking is called the RootCatalog.
Some examples

- When you request a file like http://www.someserver.com/file.html the file is found in rootCatalog/file.html
- When you request a file like http://www.someserver.com/directory/file.html is found in rootCatalog/directory/file.html

Declare your root catalog like private static readonly string RootCatalog = "c:/temp";
Useful C# API:

- System.IO.FileStream
- The method Stream.CopyTo(...)

Make sure the file stream is closed properly in a *using* or finally *statement*.
Then you can read the request send from the browser to the server and hopefully se the file vizualized.