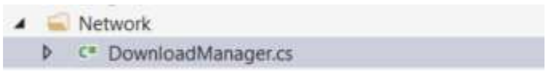
Lab: Asynchronous Programming

In our final piece of the **BashSoft** we are going to implement functionality for downloading a file from the internet using asynchronous and synchronous methods. First we are going to implement the synchronous one and after that reuse it, to make it asynchronous.

Problem 1. Start Implementing the Download Method

Let's first start by making a new folder called **Network** in our project with a new public static class called **DowloadManager**:


Network
DownloadManager.cs

…

```
public static class DownloadManager
{

}
```

All we need as information in order to download a file is the **URL** to the specified file. That's why we are going to make a public static void **Download** method with one input parameter which is the **fileURL**:

```
public static void Download(string fileURL)
{

}
```

Since we are going to make a communication with the Internet, we are going to have an API that gives us such functionality. A class that gives us easy access to the **WWW** is the class **WebClient** class which can be found in the **System.Net**. It is not a static class and for that reason we are going to have to make a new **WebClient** in our method.

```
public static void Download(string fileURL)
{
    WebClient webClient = new WebClient();
```

We can use the method **DownloadFile** from the **WebClient** that takes as a first parameter the **URL** to the image and as a second parameter, the path + file of the local machine. We can easily get the current path on the PC, but first we need to be able to extract the name of the file from the **URL**, so that we can use it to save the file with the same name on our PC.

For that reason we should make a new method called **ExtractNameOfFile** that is private static, returns a string (the actual file name) and take as a parameter the **fileURL**.

Problem 2. Extract Name of File from URL

This is how the method should look before the declaration of its implementation:

```
private static string ExtractNameOfFile(string fileURL)
{

}
```

All we need to do it take the index of the last forward slash, because it separates the file name + extension from the folder it is in, on the server. Since the result can be either a non negative number if the **fileURL** contains such a character, or **-1** if it does not, we want to make sure that we do the job only when the index is correct and otherwise throw a new **WebException** with a the message for invalid path.

```
private static string ExtractNameOfFile(string fileURL)
{
    int indexOfLastBackSlash = fileURL.LastIndexOf("/");

    if (indexOfLastBackSlash != -1)
    {

    }
    else
    {
        throw new WebException(ExceptionMessages.InvalidPath);
    }
}
```

Our method is still underlined, because we haven't returned a value for every case of our program, so let's do this and go back to the implementation of the **Download** method. All we need to do is take everything from index found + 1 till the end of the **URL** and use it as a file name.

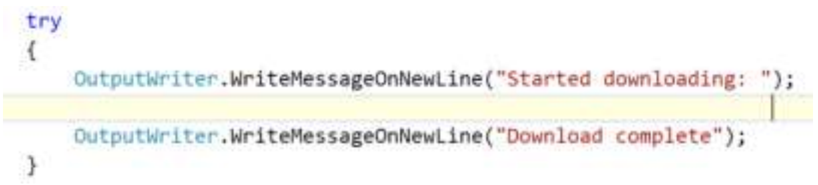
```
return fileURL.Substring(indexOfLastBackSlash + 1);
```

Problem 3. Finish Implementing the Download Method

Since we are going to use the **ExtractNameOfFile** and it can raise an exception, we would like to put all the functionality that we are going to write in a try block and in the catch block we are going to catch a web exception and display its message using the **DisplayException**.

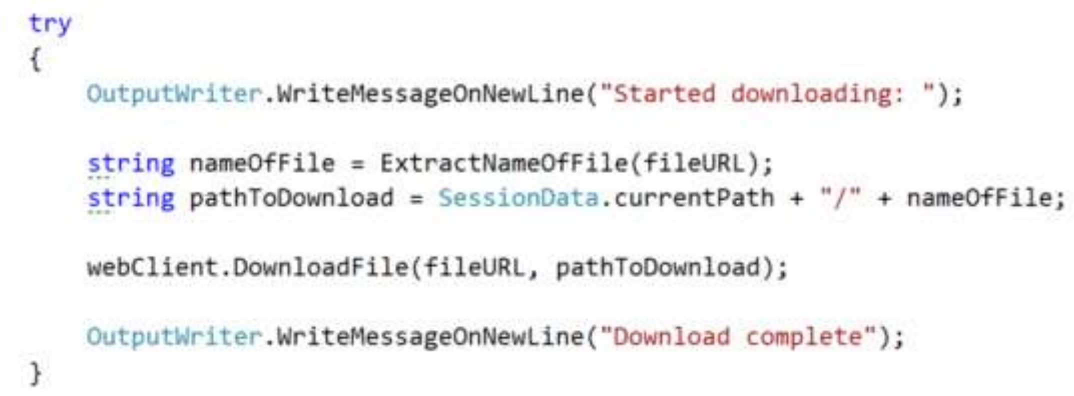
```
WebClient webClient = new WebClient();

try
{
}
catch (WebException ex)
{
    OutputWriter.DisplayException(ex.Message);
}
```

On the first line **in** the **try block** we should **indicate** the **user** that we're **starting to download** the file and **in** the **last line** in the try we should **indicate** the **user** that the **download** is **complete**.



```
try
{
    OutputWriter.WriteMessageOnNewLine("Started downloading: ");

    OutputWriter.WriteMessageOnNewLine("Download complete");
}
```

Between the two messages we should save the extracted name of the file, after that generate the local path + the name of the file and finally pass the **fileURL** and the **local path +** the **name of** the **file** to the **DownloadFile** method of the **webClient**:

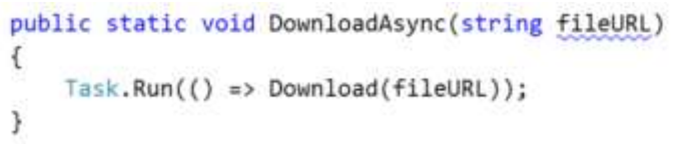


```
try
{
    OutputWriter.WriteMessageOnNewLine("Started downloading: ");

    string nameOfFile = ExtractNameOfFile(fileURL);
    string pathToDownload = SessionData.currentPath + "/" + nameOfFile;

    webClient.DownloadFile(fileURL, pathToDownload);

    OutputWriter.WriteMessageOnNewLine("Download complete");
}
```

There are other possible exceptions that a could be raised, but most of them are **WebExceptions** so we can leave it to be the only **exception cached**.

Problem 4. Implement Asynchronous Downloading

Now that we have implemented the **Download** of a file in **C#** we can easily extend it to be asynchronous, by running it on another thread as a separate task. Since the declaration of the method is pretty close to the previous, here is how it should look:

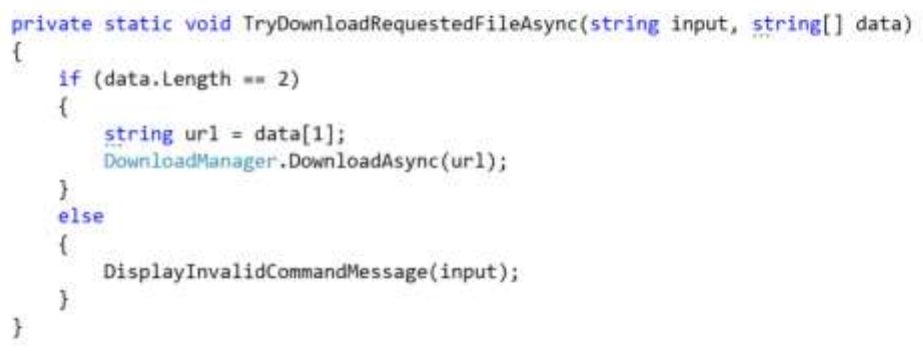


```
public static void DownloadAsync(string fileURL)
{
    Task.Run(() => Download(fileURL));
}
```

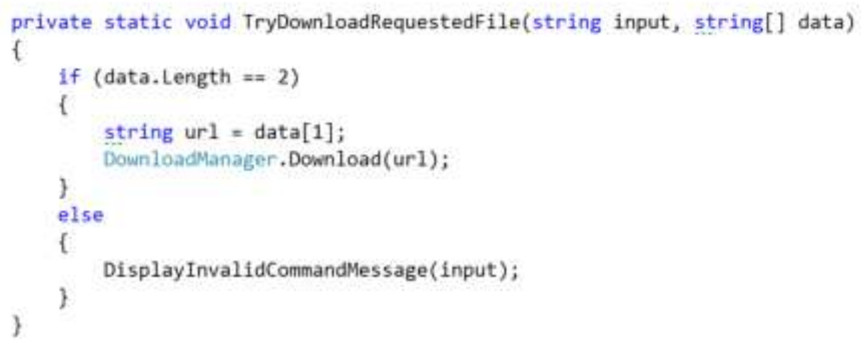
We are not going to go into depth of how the **Task** should be used properly, because that is not the most appropriate way to be used, but that needs using **OOP** to some extent.

Problem 5. Implement Calls from the Command Interpreter

The implementation of the functions that call the **DownloadManager** is pretty common to all others in the current class. We check **if** the **number of parameters** is **2** and if so, **pass** the **second parameter** to the corresponding method.



```
private static void TryDownloadRequestedFileAsync(string input, string[] data)
{
    if (data.Length == 2)
    {
        string url = data[1];
        DownloadManager.DownloadAsync(url);
    }
    else
    {
        DisplayInvalidCommandMessage(input);
    }
}
```

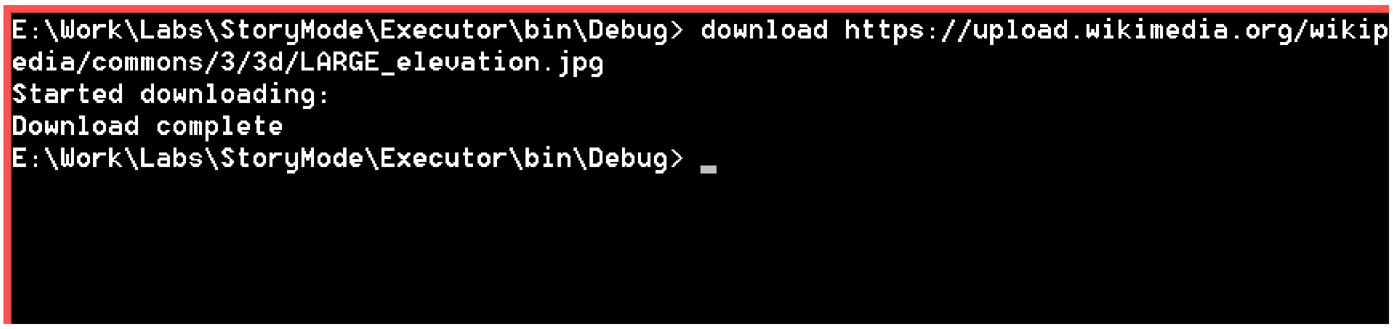


```
private static void TryDownloadRequestedFile(string input, string[] data)
{
    if (data.Length == 2)
    {
        string url = data[1];
        DownloadManager.Download(url);
    }
    else
    {
        DisplayInvalidCommandMessage(input);
    }
}
```

After that all we should do is **call them in** the **appropriate place in** the **InterpredCommand** method.

Finally let's test the functionality:

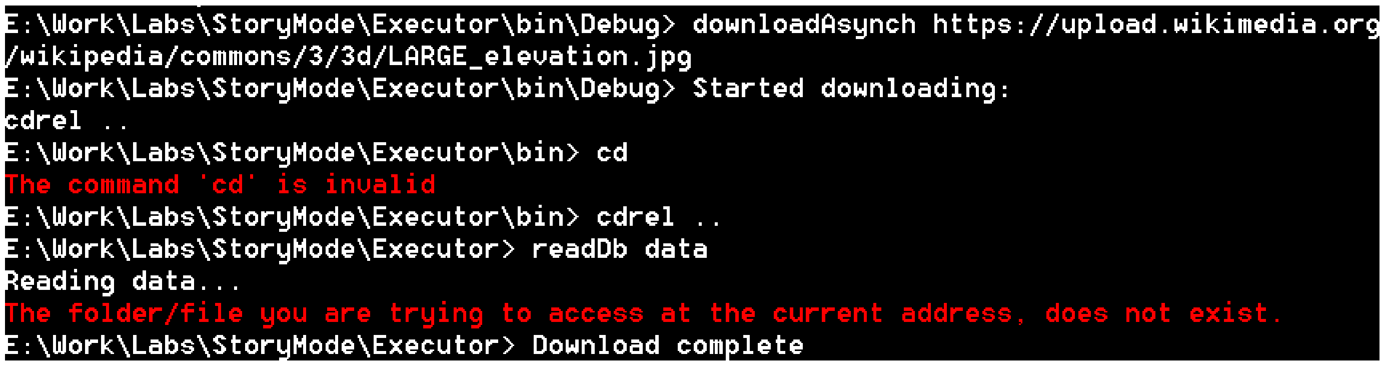
Synchronously:



```
E:\Work\Labs\StoryMode\Executor\bin\Debug> download https://upload.wikimedia.org/wikip
edia/commons/3/3d/LARGE_elevation.jpg
Started downloading:
Download complete
E:\Work\Labs\StoryMode\Executor\bin\Debug> _
```

We cannot change our position in the file system while downloading.

Asynchronously:



```
E:\Work\Labs\StoryMode\Executor\bin\Debug> downloadAsynch https://upload.wikimedia.org
/wikipedia/commons/3/3d/LARGE_elevation.jpg
E:\Work\Labs\StoryMode\Executor\bin\Debug> Started downloading:
cdrel ..
E:\Work\Labs\StoryMode\Executor\bin> cd
The command 'cd' is invalid
E:\Work\Labs\StoryMode\Executor\bin> cdrel ..
E:\Work\Labs\StoryMode\Executor> readDb data
Reading data...
The folder/file you are trying to access at the current address, does not exist.
E:\Work\Labs\StoryMode\Executor> Download complete
```

We can do much more stuff while downloading the file.

This is everything from the functionality that we are going to implement in the current course. Since many of the things that we did are quite smelly as a code, it will be our duty to fix those pieces of code and extend the functionality even more.