INFO-3137 – Project 1

Professor	Daniel Maclam
Semester	Summer 2022
Assigned	May 20, 2022
Due	June 05, 2022
Worth	20%

Your Task

You are to create an Abstract Factory for creating HTML and Markdown documents. Each concrete implementation of the Abstract Factory is to be implemented as a Singleton. As well, the document factory is to use a Factory Method to create elements in the document.

Details

The Factories:

- Need to have one concrete implementation each for both Html and Markdown documents
  - Html documents are created using the HTML5 standard
  - Markdown documents are created in the github format
    - See Markdown Cheatsheet for details
- Need to be implemented as Singletons
- Need to be able to create a concrete document implementation
- Need to be able to create multiple element types from the Factory Method "CreateElement." The types you need to be able to create are:
  - Header (h1 to h3 at least)
  - Image (including alt text and title text)
  - List (ordered and unordered)
  - Table (including table header and table rows)

The Documents

- Need to be able to add elements through a single AddElement method
- Need to be able to run the document
  - This entails saving the document text to a file
  - It also entails running a browser to display them. Some details for one way to do this are outlined later in this document.
- Accordingly, you'll need to create an interface for HTML or Markdown Documents. It should contain the following methods:
  - `void AddElement (IElement element);`
    - When implemented in concrete HTML and Markdown Document classes, this should add elements to a current list of elements.
  - `void RunDocument();`
    - When implemented in concrete HTML and Markdown Document classes, this should generate either an HTML document or a Markdown Document and open it in chrome.

The Elements

- Need to store properties and be able to output the appropriate text for their own type to be properly added to the document.
- Accordingly, you'll need an interface for HTML or Markdown Elements. This interface doesn't actually need any instance variables or methods, but will instead be implemented by individual concrete HTML and Markdown Element classes (ex. HTMLHeader, MarkdownHeader, HTMLTable, MarkdownTable, and so on)
- These concrete element classes
    - Should contain a constructor that takes and assigns the properties of the specific element
        - Note that these properties will vary depending on the element. To see what they would look like, take a look at *docs/CreateDocumentScript.txt*. For example, an HtmlImage would have properties that look like "`img/csharp.png;Alt Text;Title Text`", as specified in the .txt file.
    - Should contain a toString method that outputs a formatted string version of the properties, specific to the given document and element.
        - For example, in a concrete html image class, given the properties specified above, toString should return "<img alt='Alt Text' title='Title Text' src='img/csharp.png' /><br />"
        - Take a look at Parsing_Tutorial.pdf, for some ideas on how to parse the properties
    - In the RunDocument() method, implemented in your concrete Document classes, you'll ultimately end up calling these toString() methods. If you've implemented everything correctly, you should only have to call toString() once in each RunDocument() method (if you're not sure how, feel free to ask!)

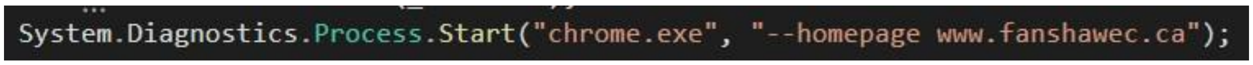
What you are provided with

Use the INFO3137-Project1 solution as a starting point. It is currently not able to compile and run, due to missing important interface types. It is your task to create the rest of the code. The solution contains:

- A project called Director
    - This project will script your document creation
    - It contains a reference to the DLL you'll use to create your factory
    - The project is also set to output to the appropriate directory to run the project
- A project called DocumentFactory
    - This is where your factory implementation will go
    - You are provided with the IDocumentFactory interface to get you started
    - You are to create all other interfaces and implementations in this project
- A folder called "docs" (not visible in Visual Studio)
    - It contains a file called CreateDocumentScript.txt. This will be used to test your factories.
        - The details of the syntax of this file will be outlined later in this document.
    - Lastly, it contains an image subfolder, where there is a demo image that is to be used in your documents.

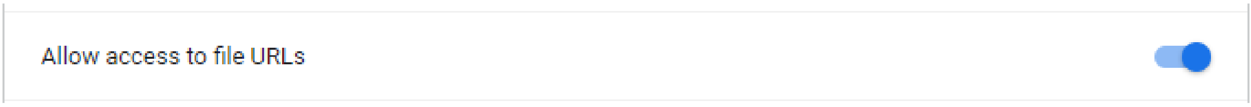
Things you'll need

- The demo project is implemented using Chrome, which means you'll need chrome installed to run it
- To run Chrome from the command line, opening to a given page, you can use the following line of code



```
System.Diagnostics.Process.Start("chrome.exe", "--homepage www.fanshawec.ca");
```

    - You'll still need to get the absolute local file path for the file you intend to open.
- To view the Markdown document formatted properly in the browser, you'll need this extension
    - To allow the extension to read local markdown files, you'll need to go in to the extension settings and set "Allow access to file URLs" to true



Allow access to file URLs

The CreateDocumentScript.txt Format

Some of the processing of this file is already handled for you, but you'll still need to do some string processing to get the various properties out of the file.

- Each "line" of the file is considered to end when the character '#' is read
- The first part of each "line" is one of the following, and is delimited by a ':'
    - Document
    - Element
    - Run
- The Document line contains two sub parts, delimited by the ';' character
    - The type
    - The file name
    - You'll use these to create a factory and a document
- The Element line contains:
    - The element type, followed by a ':'
    - This properties for the element
- The Table Element has some special syntax
    - Each "line" of the table is delimited by a ';'
    - Each of the parts of the table is delimited by a '$'
    - The first item in each "line" is the type of the table row you're creating, either header or row
- The Run line contains only the word run. This will create and run the actual document on disk

Grade Breakdown

Detail	**Points**	**%**
Properly implement IDocument Factory for • Html • Markdown	 **10** **10**	 **10** **10**
Factories are implemented as Singletons		

• Html • Markdown	**5** **5**	**5** **5**
Element creation is handled through the factory method • Html • Markdown	 **10** **10**	 **10** **10**
Factory methods handle the creation of: • Markdown Header • Markdown Image • Markdown List • Markdown Table • Html Header • Html Image • Html List • Html Table	 **5** **5** **5** **5** **5** **5** **5** **5**	 **5** **5** **5** **5** **5** **5** **5** **5**
Document save to disk and opens in a browser • Markdown • Html	 **5** **5**	 **5** **5**
Total:	**100**	**100**