

Design Document

The architecture of our library makes use of the abstract factory, the composite and the visitor pattern. In this document, these three design patterns will firstly be illustrated by UML with multiple classes inside as example, and then we will further explain how they can benefit our library.

General Class Structure

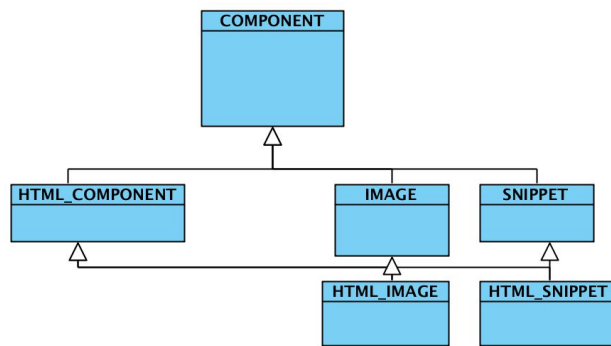


Figure 1 General Structure UML

For each class in the API will follow the structure illustrated above. It will inherit from either Markdown or HTML component and its own deferred class. This design allow easy extensions of MARKDOWN_COMPONENT in the future. It can be simply implemented by adding classes inheriting from MARKDOWN_COMPONENT and the deferred class.

Abstract Factory Pattern

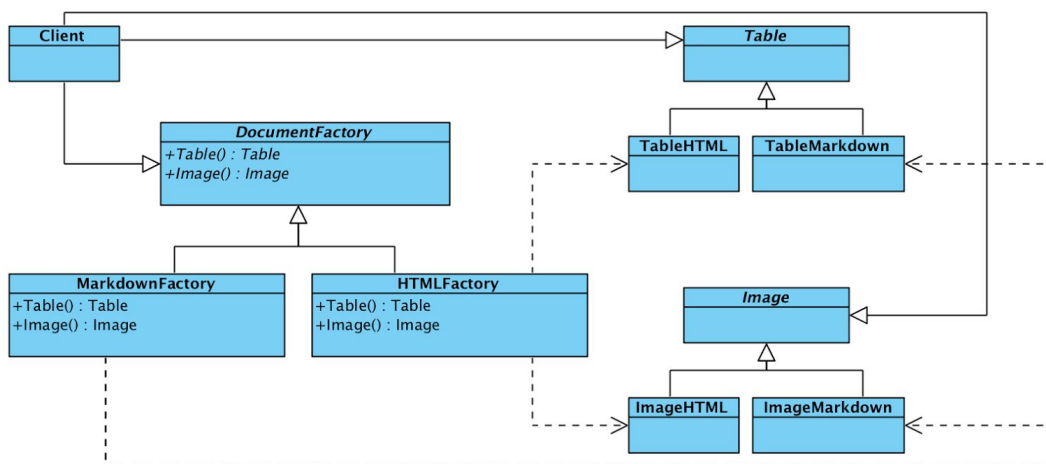


Figure 2 Abstract Factory UML

Support for HTML and Markdown

We used the abstract factory pattern to keep our program flexible so we can easily extend our library to markdown. So far, our plan is to implement only the HTML_Factory unless we have spare time. To initialize the Markdown_Factory one just has to implement the same set of concrete classes as for the HTML_Factory. Once both Factories have been implemented, a user can switch very easy

between the two output versions. By sticking to the abstract factory pattern, we can ensure that the factories do not mix in between. In this way we can guarantee that we either get a pure HTML or a pure Markdown document.

Composite Pattern

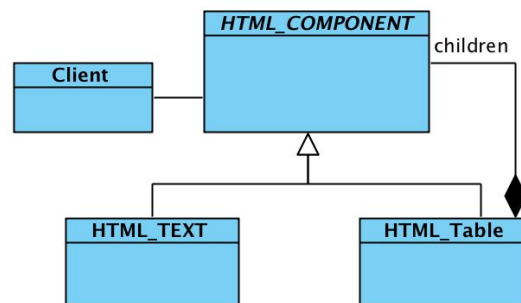


Figure 3 Composite Pattern UML

Organization of the text elements

Our library supports encapsulation, which means for example that you can place a bold text and an image inside of a table.


Name	Picture
Hans Mustermann	

Figure 4 Example of Table Composite

To implement this, we used the constructor pattern. All classes of the constructor pattern can be dedicated to one of two groups.

The classes `table`, `list` and `paragraph` are composite classes. As arguments they can take simple text or any class of the `HTML_Factory`. Theoretically, one can encapsulate an infinite number of classes, but we assume out of imaginable use cases that there will be no more than three encapsulations. In our views it is still user-friendly to ask the user to write three encapsulations in one row.

The classes `image`, `text`, `title`, `link` and `snippet` are leaf classes and cannot take any other class as input. They accept only a string as input and depending on the class some integer parameter as well.

Visitor Pattern

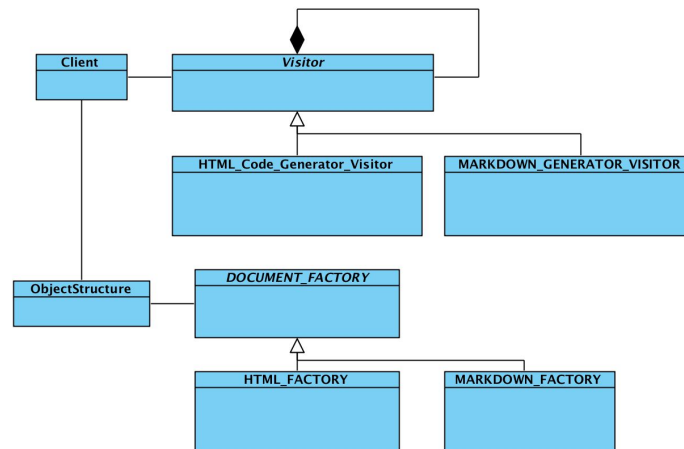


Figure 4 Visitor Pattern UML

Code Generating

The classes of the HTML_Factory are not doing anything except of generating objects and it inherits DOCUMENT_FACTORY, which is the deferred class for both HTML and Markdown modes. In our API, we only implement HTML_Code_Generator_Visitor, which will be responsible for generating the corresponding HTML code for each objects . It visits all the generated objects in the right order for the list and extracts the HTML code that is stored inside the object. Finally it puts all that code together and stores it in a string.

Other Features

User input

The first thing the user does is to list all elements the user wants to create and define their types. Then he can create the elements and add their content. Finally the user adds the over element/s that should be displayed to the list. The user input for the example showed before will then look for example like this:

text1:TEXT

text2:TEXT

text3:TEXT

image:IMAGE

table:TABLE

text1:=html.text("name")

text1.bold

text2:=html.text("picture")

text2.bold()

text3:=html.text("Hans Mustermann")

image:=html.image("C:/user/pictures...", "Profile Picture")

```
table:=html.table(2)
table.add_row([text1,text2])
table.add_row([text3, image])
```

```
list.extend(table)
```

In this example it would still be possible to hand over the input directly as an argument, but as soon as the input gets longer the last line would end up as a messy monster.

Changes in the Design Document

14.11.2017

The user input had to be changed, the user no longer has any freedom to choose whether he wants to assign a variable to it but has to in order to make the visitor work. Therefore, the user inputs examples also had to be changed.

04.12.2017

Updated diagrams and user input.