Quicktag

A simple web framework in Java

Olav Aleksander Nøklestad

Group 40

Table of Contents

Introduction

The goal is to make a web framework with Java as the primary programming language. The idea behind the framework is that it will generate the appropriate HTML and CSS files to set up a website, based on the client code you write in the program that actually uses the framework. This could involve options such as selecting the size of the grid layout on the page, or selecting different premade themes based on what kind of website you want. Certain standard objects or widgets like buttons, menus and a navbar is also something that should be possible to add by declaring these objects in the client code when using the framework. Making changes like adding a button should also be easy enough, as the HTML and CSS files would be quickly generated again based on your client code.

When designing the API, I wanted to prioritize ease of use. I think it’s important to give the user a reason to actually use the API, so that they won’t go back to manually writing HTML and CSS instead. In other words the API should ideally allow you to set up a website both faster and easier compared to simply writing HTML and CSS the standard way.

The basic idea of the web API is that you first declare or initialize objects/elements (HTML), and then style those elements with CSS. I had first planned to go for an object based method, where you would make a paragraph object for instance, by sending some parameters to a paragraph constructor. When testing this approach I found it clunky; everytime you want to make a new object, you would also have to give it an appropriate variable type and name for every object you make. I found this object based approach to require more from the user than I would like, as it is up to the user to initialize the proper variables for every new object.

Instead I went for a method that didn’t require you to make objects or variables. To make a paragraph for instance, you can simply call on the “p” method, and pass some text through as a parameter and that’s it. The string value will then be stored in a global variable, which will then be written to a HTML file when the user calls a “Write” method at the end. With this approach, the user doesn’t have to initialize or make many objects or variables, but simply call on a method and send in the appropriate data to make HTML elements.

Ultimately I like this approach better compared to making objects with a class-constructor or builder pattern because it means less work and less typing for the user. Similarly I also wanted to make styling easy to do, and keep everything related to one element on as few lines as possible.

If the user calls on a styling method right after the line of a “Basic” method, they can style HTML elements without even giving them an ID. The CSS-method will instead fetch the generated ID of the recently created element, and use that ID to style it. If you give the element an ID however, there are “Style” methods that allow you to target elements based on ID as well.

Background

Since I decided that I would make my web framework primarily in Java, I looked for existing solutions that were also Java-based. With that in mind, a framework called Spring is arguably the most popular web framework, so I thought it could be useful to see what could be done with the use of Spring.

Spring however seemed to be more application based, and didn't really function the way that I thought it would. Generally you will make your web page using HTML and CSS like normal, but Spring provides various features that you can *combine* with your already existing website to make it more application-based. This allows you to make websites with a lot of functionality, but is not really the kind of framework that I had intended to make.

I looked for some other web frameworks, and Ruby on Rails seemed like another popular choice, but just not Java-based. This framework is still however a bit closer to what I had imagined that these kinds of web frameworks would provide, but still quite different. There are some things that are simplified here in regard to making HTML creation easier and faster, but the focus of the framework seems to be more similar to that of Spring. In other words, the general idea is to provide a framework where you can easily integrate more complicated functionality to your web page. This means that it doesn't look like Ruby on Rails attempts to streamline HTML and CSS creation like I attempt to do with my own framework.

When it comes to front end only, there is a very popular framework called Bootstrap, though it's not related to programming. It simply streamlines the process of styling your page by allowing you to access a big library of very organized CSS styling methods.

In general it seemed that most web frameworks focus on integrating programming functionality into web pages, which is different from the kind of framework I had set out to make.

Method

API Design Specification

When working on the project, what I had in mind most of the time was to make user scenarios that were easy to use, and I really wanted my framework to reflect that. I did a lot of experimentation in Java to figure out how I could pass around parameters and how to make "objects" in different ways. A lot of focus was also put into finding a solution to make HTML elements with the least amount of user input required, and my scenarios were eventually changed based on my findings.

To minimize user input as much as possible for my framework, I also came to the conclusion that I would have to get rid of as much object creation as possible, and that was also part of my primary focus. The bare minimum of input that I was able to achieve was for the user to simply pass through their desired values, and an "object" would be made. Ultimately I just didn't want to make a framework that required the user to make a ton of different objects and make a bunch of initializations just to get things done.

Design Patterns

For the most part, the framework was developed almost entirely without design patterns in mind, though I did take a few into consideration along the way. After looking into several design patterns, I found that many of them were very object-heavy or object-based. Builder pattern for instance would give a flexible structure to object building where you could provide an object with only the properties you want. On the other hand this kind of pattern would greatly increase the amount of objects you would have to make for each HTML element as well as all the property methods you have to call for each object.

I found similar issues with other design patterns that they were either too object-based or just not really needed for the type of framework that I at least had in mind. That’s why for the most part the framework was made with the user scenarios in mind, and not really any particular design patterns.

Implementation

The general strategy I went with to actually make a working implementation of the framework was more or less the same for every method. A method would take some input values from the user, and then append those values to a storage variable, encapsulated with the relevant HTML tag. So for a paragraph method for instance, the method would just store something like this in a general variable: <p>Text from user!</p>. When calling on a “write” method, all the content stored in the storage variables would be written to HTML and CSS pages.

User Testing Strategy

As my main focus all this time had been user-friendliness and minimal user input, I was mostly curious about how first time users would respond to making some of the more “basic” elements. For the user testing I had prepared five different user scenarios or tasks that I would ask the testing group to complete. I also prepared a quick start guide that was meant to help with introducing testers to the framework, but I also planned to be available during testing if more help was needed.

The five tasks I gave were the following:

1. Make a paragraph object (p)
2. Insert an image (img)
3. Make a paragraph object with a unique ID
4. Give the paragraph object some styling (text color, font size etc)
5. Make a navbar (nav)

I had structured the tasks so that they would hopefully be easy at the start, and then slowly escalate to some trickier methods. I anticipated that the main takeaway from the user testing would be how each tester interacted with my framework, and I would then try to make adjustments accordingly if someone did not find it to be user-friendly.

The Design Process / Results

Changes in Specification

In regard to design specification and user scenarios, there were some drastic changes made along the way compared to the very first pseudo code I made. My very first pseudo code for making a paragraph element and styling it looked like this:

```
public static Framework.Text newParagraph = new Framework.Text();

newParagraph.color("green");
newParagraph.text("Here you can write whatever you want!");
newParagraph.size(20);
```

This was essentially a very object driven approach, where you would have to make a new object (newParagraph) for every single HTML element you wanted. The idea at the time was that the user could then apply properties and attributes to these objects, like color or text size. When I experimented with this approach in Java, I quickly came to realize that I didn’t really like the aspect of making an object for every single HTML element and that it felt somewhat clunky as a user to have to initialize an object every time.

HTML and CSS are already inherently very straightforward, easy to use and quick to get started with. That’s why I didn’t want to end up with a framework that would simply be more tedious and more bothersome to use than simply just writing HTML and CSS manually. With these things in mind I changed my focus quite early on to look for a structure that essentially would have no object creation at all, only method calling. To highlight the difference, below is the same example as shown above, but instead with the pseudo code I later on aspired to implement.

```
Web.p(“Here you can write whatever you want!”);
Web.css(“color: green;”, “font-size: 20px;”);
```

The final implementation is different from this, but a lot closer compared to the very early pseudo code.

Changes in Design

For the design in particular, there were a few changes made once I moved away from the object based idea that I first had. The general idea at the time was to make methods that would append HTML or CSS text to global storage variables. You would then call on a method that would write all that stored content onto HTML and CSS files. This idea would remain mostly the same until the very end, though some bigger changes were made for the final delivery.

At first I had put everything in one class “Framework” intentionally to lessen the amount of things the user would have to instantiate. An interface called “Controller” was then used to make an instance (called “web”) of this class, which could then be used to make everything the framework had to offer.

Originally I also had an idea to implement method chaining, which is why most of my methods were class methods (public Framework) instead of void to begin with. This focus on method chaining mostly came from a misunderstanding on my part on how it actually worked. Basically I thought that method chaining could improve readability in code when you would have many elements with styling. Some of my earlier HTML methods would store a temporary ID in a storage variable, that would then be picked up by a CSS-styling method to style the most recently created element. I thought first that this only worked because of method chaining in Java, but I came to realize that it would still work if a styling method was called on the next line. I assume method chaining can modify values immediately if you are actually “returning” values in subsequent methods. In my methods however, I was simply storing an ID temporarily in a variable and not returning anything, so method chaining was pointless. In the end I just made all my methods void instead, as they weren’t supposed to return anything anyway.

All the storage variables used by the methods were also just placed inside the same Framework class, and ended up being a little confusing to deal with as I was often working with more variables than I really needed. The final implementation for my data storage would instead keep the amount of variables to a minimum, make them private, and also have them “centralized” in a separate class “Data”. This made data management more structured and also made it easier to debug things, as data handling would now be separate from the methods that made the HTML elements.

Changes in Implementation

One big change from my earlier versions of the implementation was classes and general structure. As mentioned previously, I first had all my methods in a single class “Framework” to lessen the amount of object creation for users. After some consideration based on user feedback however, I did eventually end up abstracting out my methods into what was hopefully more appropriate classes. I made new classes based on some common HTML categories, and then placed methods around into where I thought they would best fit. I then divided all the classes into some relevant packages: html, CSS and data.

The data package would include the main class “Data” for handling HTML/CSS information and the interface “Controller” that would give an access point to the class. All the other classes would implement this interface so that they could use the methods in “Data” to add and remove html elements and styling. When I started adding more classes, I thought it would be messy to have separate storage variables in every class, and that’s why I decided to keep them all contained in the “Data” class instead.

User Testing Feedback

First Iteration

Tested by: Group 12

My first round of user testing was done with project group 12. I was mainly curious to see how they would perform when it came to making elements/objects, as I have been trying to make a framework that aims to be user-friendly. To perform the testing, I handed them a quick start guide. They were also given five different tasks, or scenarios, that were similar to my original design scenarios and mainly revolved around making different kinds of objects. The five tasks given were the ones mentioned in User Testing Strategy.

I was also present over voice chat to assist them if they needed. The tasks were completed fairly quickly with not much assistance asked for. Here is the following feedback I received from each member:

Mats

Veldig bra at man ikke instansierer mange objekter, kan være lurt å skille noen ting inn i klasser.
Implementasjon fungerer, dette er veldig bra, husk at dette bør være skjermet fra bruker(private, protected). Hva tenker du skal skille høy-nivå og lav-nivå api-ene i dette rammeverket? Husk DRY-kode. Hva slags design-pattern bruker du?

Håkon

Første inntrykket er ganske så overveldede med antall linjer med kode i samme klasse (class Frameword). Det vil ikke ha så mye å si når man bruker det til å skrive koden, men gjør det tung å lære seg.

Heading delen (starten) av koden er ganske tung og over velmenende.
eks…

```
public Framework h1 (String text){
   String input = "<h1>" + text + "</h1>";
   storage += input;

   return this;
}

public Framework h1 (String id, String text){
   String input = "<h1 id=\"" + id + "\">"  + text + "</h1>";
   storage += input;

   return this;
}
```

For meg så føles det bedre ut hvis du hadde skrevet…

```
Public Framwork h1 (String text, Sting id){}
```

Og/eller skrevet den noe som dette

```
Public Framwork heading (int lvl ,String text, Sting id){
```

String input = «<h» + lvl + «>» + …… + «</h» + lvl + «>»
}
og med dette så kan dette bli letere å skrive koden og det blir mindre å huske.
Hvis dette skulle bli tatt i bruk så tipper jeg på at du kan fjerne rundt 100 linjer med kode.

Rundt linje 180 til 210 så kan man lage div'er, main, osv. men jeg forsto ikke helt hvordan dette fungerer, i selve Framework.java og når jeg skal prøvde å lage koden.

Jeg kom ikke helt til CSS delen (det å eksperimentere så mye med den), men jeg har lest koden og jeg må si den føles veldig tvunget.
Dette er fra linje 560
public void css_text(String color, String alignment, String fontFamily, String fontStyle, int fontSize){

```
   if(!color.isEmpty()){
      output += "color: " + color + ";\n";
   }
   if(!alignment.isEmpty()){
      output += "text-align: " + alignment + ";\n";
   }
   if(!fontFamily.isEmpty()){
      output += "font-family: " + fontFamily + ";\n";
   }
   if(!fontStyle.isEmpty()){
      output += "font-style: " + fontStyle + ";\n";
   }
   if(fontSize != 0){
      output += "font-size: " + fontSize + "px;\n";
   }

   if(id.equals("")) {
      style += "#" + finalId + " {\n" + output + "}\n\n";
   }
   else{
      style += "#" + id + " {\n" + output + "}\n\n";
   }
   output = "";
   id = "";
}
```

Jeg får en feil som sørger for at jeg ikke får teste ut mer, men når jeg jobba med dette så ville den at jeg skulle fylle ut alt som var i den, selv om jeg ikke ville ha enkelte av dem med. Dette kan være en feil du eller jeg har gjort.

ALT I ALT
- Det er vanskelig å si at dette er nybegynner vennlig, men dette ser ikke ut som noe for de profesjonelle
- Kanskje litt for mye på enn side (Framework.java)

o Mange linjer med kode
o Litt sånn fram og tilbake jobbing -> eks. «heading» med id og text
o Noen av navene gir ikke mening med det første -> eks. css_text
- Når man skriver alt og skal sammenligne med generell HTML-kode så gir det mening på struktur og logikk, og dette vil gjør det letter for folk som har HTML kunnskaper å lære seg det du har laget.

Lars

From the get-go, I really like the simpleness of the framework and that it follows a clear pattern. This meant that it didn't take long for me to understand how I should make my website, and it was very hard to misuse. The one-to-one naming convention further established this, so everything was very familiar. Below I wrote some things I think should be fixed, hopefully it is useful advice.

I feel like the framework can quickly exceed the recommended method count in a class. I recommend following the rule of 30: "A class should contain an average of less than 30 methods" (https://dzone.com/articles/rule-30-%E2%80%93-when-method-class-or#:~:text=b)%20A%20class%20should%20contain,to%20900%20lines%20of%20code). Maybe group the methods into classes?

The main class should be omitted, as it's not needed in a framework.

I think the way parameters are used to determine data is not ideal, as it greatly limits what the user is able to do with the framework. And if more parameters are added, the code would get messy very fast.

There is very little layering in the code, and no real distinction between a low-level and high-level APIs here. This is bad because now it's not hiding the implementation from the user, and it may be difficult to maintain.

Try to think of a more appealing name than just "Framework".

The way interfaces are used is a bit strange. It's nice that I can simply type web.p("helloWorld"); to make a paragraph, but it can be achieved the same thing by using static methods. Maybe there could be a class named "Compiler" that handles the HTML strings and uses static methods like so:

Compiler.java

```
public class Compiler{
public static void addHtml(String html) {
// Do stuff...
}
}
Web.java
public class Web {
public static void p (String text) {
```

```
Compiler.addHtml("<p>" + text + "</p>");
}
}
```

Now in Main.java, I can call Web.p("helloWorld"); like normal.

I recommend writing JavaDoc to explain the methods more clearly to the user. It's easy, just type /** + ENTER and intellij will generate a JavaDoc comment. It also allows auto-generating Type Reference Documents later on.

Consider using StringBuilder.

The CSS methods confuse me, because three of them have an identical signature, but they have different names. The only difference between them is the way the selector is generated. Here the symbol before the selector could simply be removed, and then the user can decide whether it is a class-, id- or tag-selector:

```
public void css_rule(String selector, String... args){
String freeOutput = String.join("\n", args);
style += selector + " {\n" + freeOutput + "\n}\n\n";
}
```

I could type Style.css_rule(".myClassName", "color: blue;"); and achieve the same thing as with the css_class method.

Thoughts

Overall I was quite pleased with the feedback given from group 12, as they took their time to provide a lot of useful insight and thoughts that are easy to overlook when you work alone. I gathered all the feedback, and then tried to collect and compress them into smaller statements in the form of "tasks" that I would try to implement. Not every piece of feedback did I deem feasible to implement, but I tried to filter it the best I could for the final implementation and ultimately ended up following quite a few of their tips. The exact changes are outlined in the final section of User Testing.

I thought it was great that 2/3 members found the framework easy to use and get started with, but it also made me think about how people can expect different things from a framework. I had always assumed that having minimal object creation would always make things simpler to grasp, but I started thinking that others might actually expect some form of object creation and might find it confusing to only call methods to create things. I feel like my final implementation might accommodate for both sides a little better, but ultimately I still really wanted to stick with the plan of minimizing object creation as it had been my goal almost the entire way.

Second Iteration
Tested by: Christoffer Nøklestad

For the second iteration of testing, I asked my brother who is a former IT-student at HIOF, to test the framework. I was interested in seeing how well a more "general" user would be able to use the framework, compared to the other user groups in our project. A user that is not part of our project groups and field of study could provide some insight that the more "specialized" users might overlook. Students in the other groups will most likely understand my framework a lot faster than the average user, because they have more recent knowledge that is relevant to frameworks in general, or they are making a similar web framework themselves. This kind of user will have an easier time experimenting with the framework, and often will just “know” what to look for. In contrast, a general user might have a very different way of looking at the framework, and might struggle with parts that more specialized users find to be easy.

To carry out the test, my brother was handed the same quick start guide as group 12, and asked to perform the same five tasks with the framework, and could ask for my help if needed.

Christoffer

Flere eksempler på de ulike metodene hadde vært nyttig + bilder/screenshots. Ikke så lett noen ganger å vite hva som er riktig input

Fokus på god dokumentasjon for hver metode

Litt rotete at alt heter “web” som f.eks web.p(“Skriv noe her …”);
Spesielt når det blir mange objekter – litt vanskelig å se hva som er hva

Thoughts

Overall he didn’t have too much trouble with completing the tasks, though I did notice he would often look for documentation for a method or try to figure out how it worked. At the time my JavaDoc was not finished, but it did remind me of the importance of documenting each method properly. With this feedback in mind I would also try to make some more elaborate examples for different methods with screenshots as well. The actual changes I have outlined in the last section for user testing.

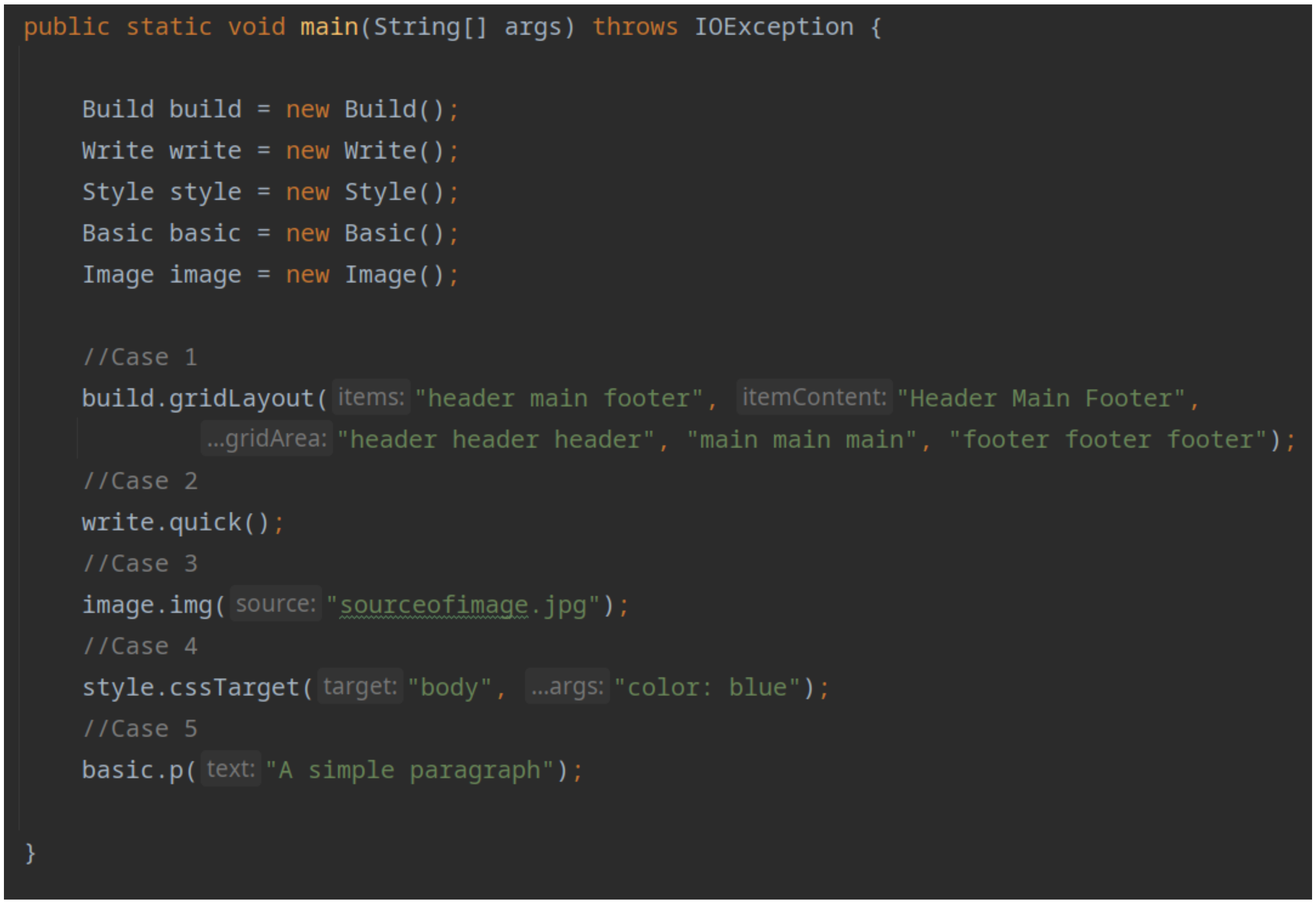
Resulting Framework

The Main Scenarios

Based on some feedback early on, I tried to make my scenarios more generalized, and so I ended up making the following cases:

1. Make a grid or flexbox layout
2. Add new HTML / CSS pages and link them
3. Add new objects
4. Select a theme for your website (styling)
5. Add content like text or images

There are multiple ways to solve these tasks with the final framework, but below is a possible solution:



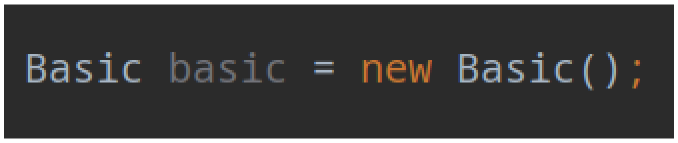
```java
public static void main(String[] args) throws IOException {

    Build build = new Build();
    Write write = new Write();
    Style style = new Style();
    Basic basic = new Basic();
    Image image = new Image();

    //Case 1
    build.gridLayout( items: "header main footer",  itemContent: "Header Main Footer",
            ...gridArea: "header header header", "main main main", "footer footer footer");
    //Case 2
    write.quick();
    //Case 3
    image.img( source: "sourceofimage.jpg");
    //Case 4
    style.cssTarget( target: "body",  ...args: "color: blue");
    //Case 5
    basic.p( text: "A simple paragraph");

}
```

Note that “Write” methods have to be called at the end of each desired HTML page. Running this would only write the grid layout to a page, and not the rest.

General Explanation

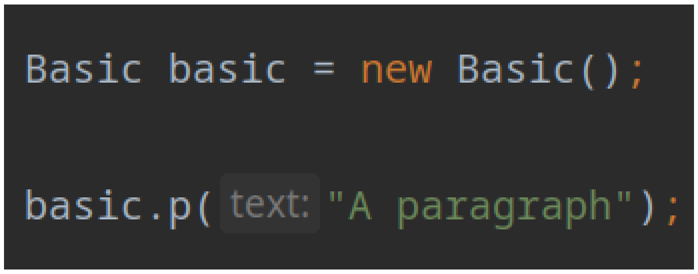
To give a very general idea of how the final framework behaves and functions, I have made an example with only a paragraph element.

1. The user makes a single class object for a class whose methods they wish to use
    1. This just acts as a reference point to access the methods, and you only need one object for each class.



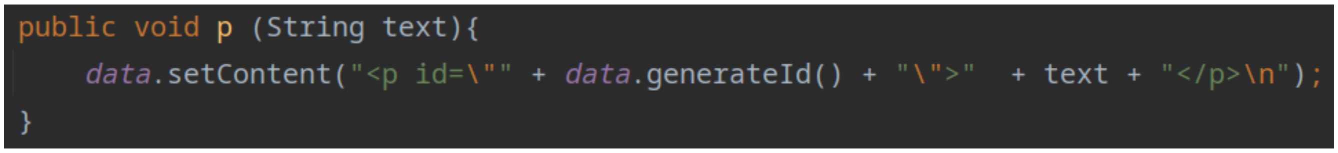
```java
Basic basic = new Basic();
```

2. The user provides a method with the desired input (method “p” for a paragraph)



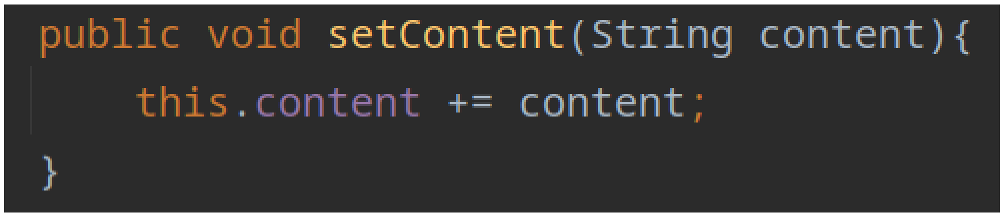
```
Basic basic = new Basic();

basic.p( text: "A paragraph");
```

3. The method then calls on another method that appends the data to a variable



```
public void p (String text){
    data.setContent("<p id=\"" + data.generateId() + "\">"  + text + "</p>\n");
}
```

4. The text string is then appended to the storage variable “content” in the “Data” class



```
public void setContent(String content){
    this.content += content;
}
```

As mentioned in the JavaDoc, “content” stores all the HTML data.

If the user also calls on a method from “Write”, the method will write all the stored data in “content” and “styling” onto generated index/CSS files. This is in a nutshell how data is stored, and then written into appropriate files.

Implementation of Design

Most of the creational design patterns seemed quite object based, and as I mentioned earlier I didn’t find any of them to be necessary for what I set out to make. I looked into builder pattern, but it would require me to essentially abstract out every single method to their own class, and then go back to making objects for each element, which is something I didn’t want. Another thing I had considered was implementing the singleton pattern. When using the different methods in my framework, you only need to initialize a class object once for each class, so I could have technically used the singleton pattern here. After doing a bit of research on it however, it seemed that opinions were very split on this pattern. Ultimately I came to the conclusion that if I don’t technically need it for my envisioned framework, then I should probably not implement it.

It’s possible that I could have developed the framework with some more structural or behavioral patterns in mind, but my primary focus for the most part was to make something that was minimal in its input and typing, and then hopefully user-friendly as well.

Implementation of User Feedback

I made various changed based on the feedback I received from my two user testing sessions. Here is a list of the changes I was able to implement for the final delivery:

- Abstract methods out into multiple classes instead of having one class
- Tried to think of a more interesting name than “Framework”
- Revised the way that data is being handled and stored
- Compressed similar methods into one to reduce redundancy (h1-h6 elements)
- Looked into using StringBuilder instead of concatenation in loops (+=), but was able to circumvent this anyway thanks to the new data structure in the final implementation
- Added examples with screenshots included on how to make various elements
- Class abstraction hopefully alleviated some of the possible confusion that could happen when every single thing was called from “web”.
- Made storage variables private to hopefully “hide” the implementation a bit
- Implemented and fully fleshed out the JavaDoc for each of the methods
- Reworked a CSS method in “Style” to allow you to choose the styling target instead (removes redundant code)
- Attempted to make a distinction between some low level and high level methods by keeping them contained in separate classes (Basic – Build)

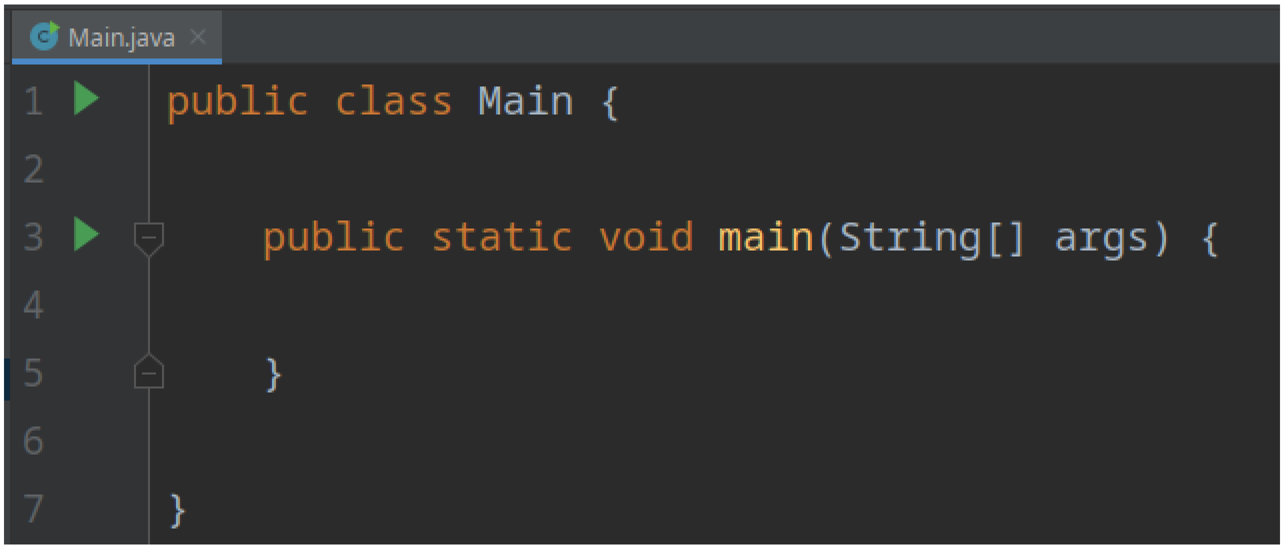
For the most part these were the more major changes I implemented in regard to the user feedback, though the feedback was also helpful when it came to more general principles as well like rule of 30 and DRY. My revised data structure and addition of new classes was an attempt to follow some of these principles as well.

Getting Started

Importing the framework

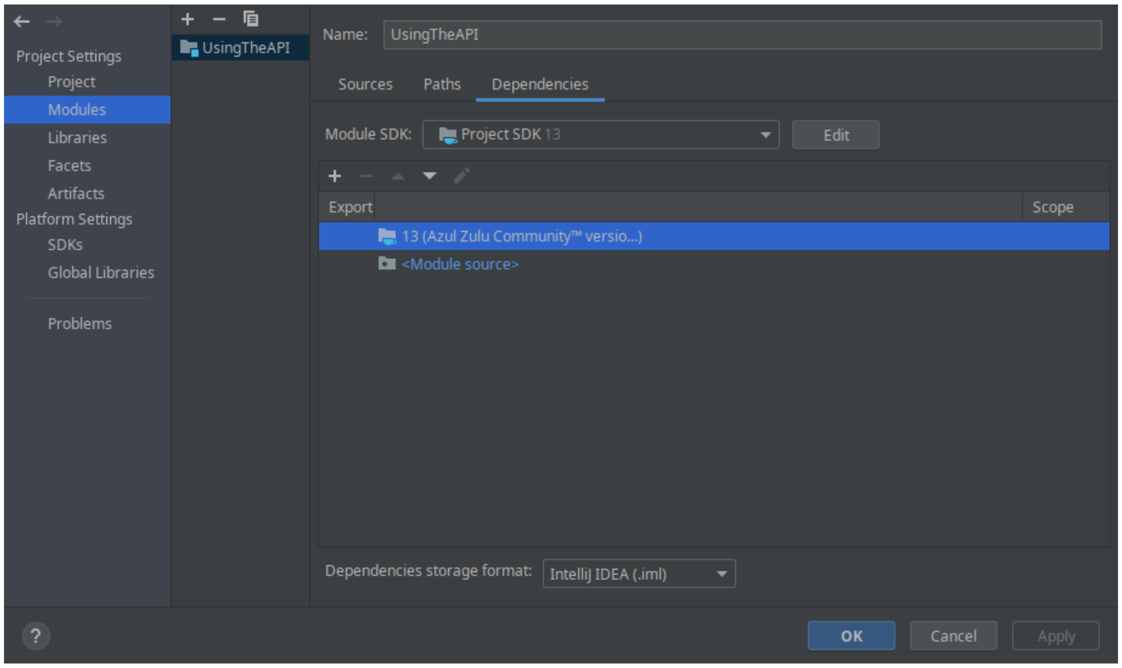
To get started and to use the framework like an actual library or API, you can start by making a new Java project in your IDE of choice. For this example I will be using IntelliJ to import the framework into a brand new project, ready to be used.

I begin by making a new Java class “Main”, and implement the standard main method so we have an area we can use for testing.



```java
public class Main {

    public static void main(String[] args) {

    }

}
```

Next, we want to actually import the JAR file into our new project. Open the Modules window by going here: **File > Project Structure > Modules**



Name: UsingTheAPI

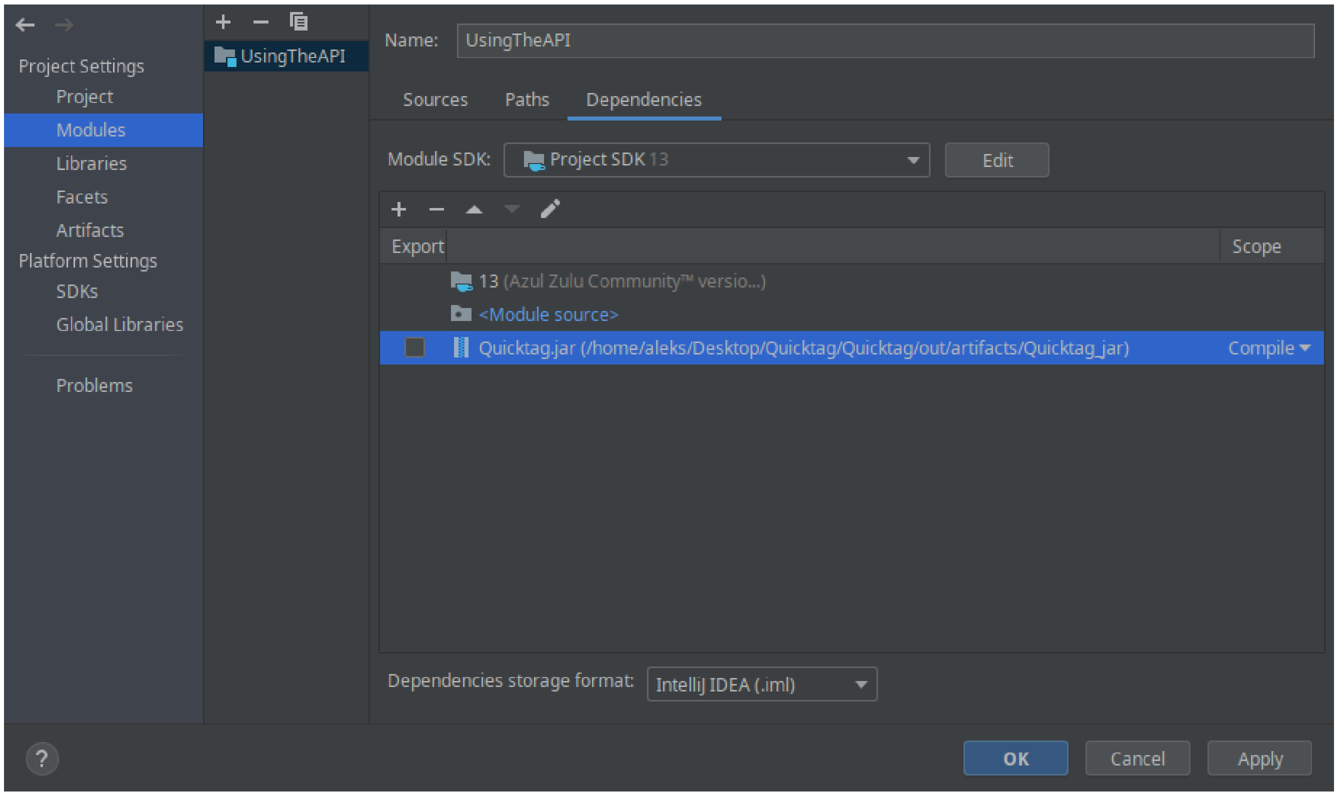
Module SDK: Project SDK 13

13 (Azul Zulu Community™ versio...)

<Module source>

Dependencies storage format: IntelliJ IDEA (.iml)

With this window open, you can press the "+" symbol above export, and then click "JARs or Directories …". Find the "Quicktag.jar" file, and press OK (path to the JAR in the project folder should be Quicktag/out/artifacts/Quicktag_jar/Quicktag.jar). Your window should look like this:



Name: UsingTheAPI

Module SDK: Project SDK 13

13 (Azul Zulu Community™ versio...)
<Module source>
Quicktag.jar (/home/aleks/Desktop/Quicktag/Quicktag/out/artifacts/Quicktag_jar) Compile

Dependencies storage format: IntelliJ IDEA (.iml)

Press OK, and Quicktag.jar should now be in your “External Libraries”.

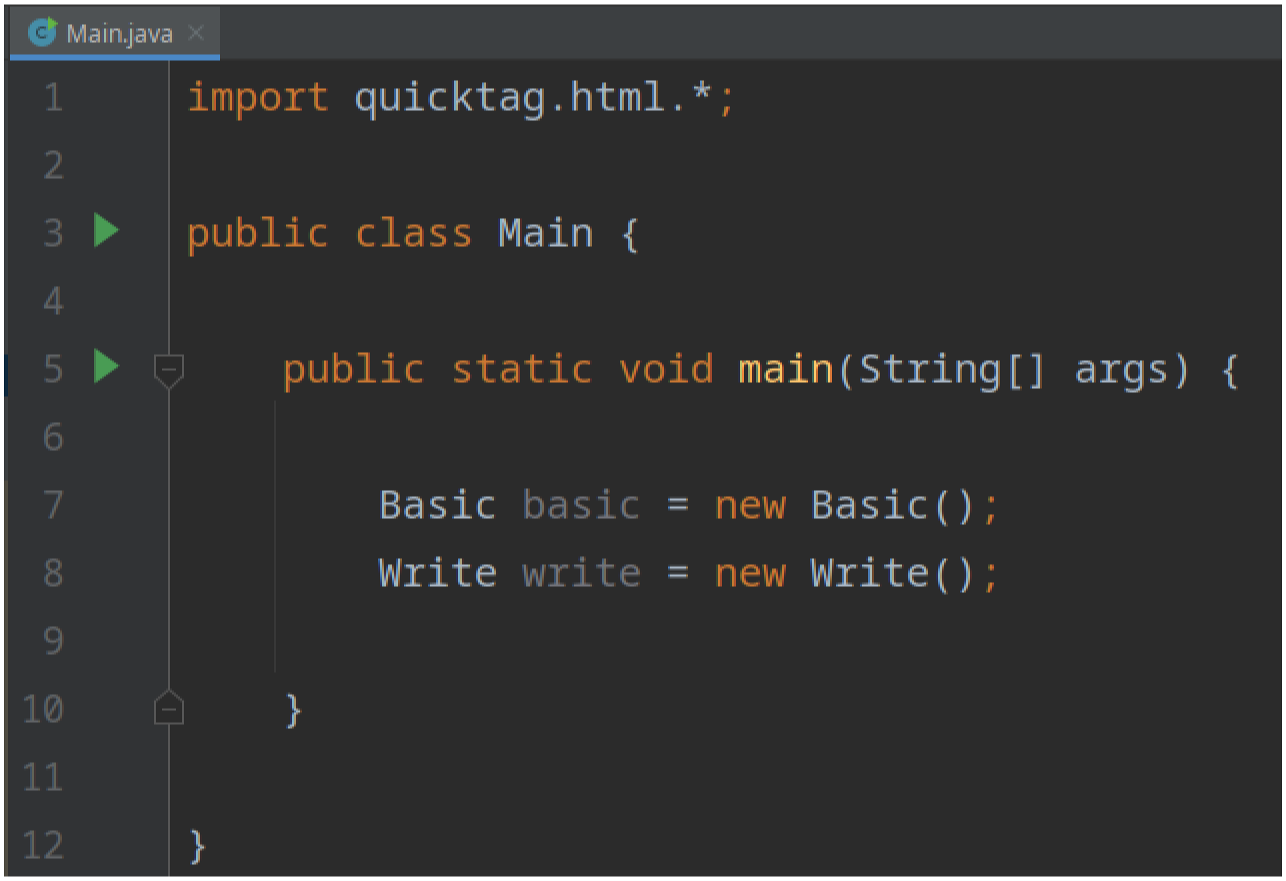
That’s it! We are now all done, and the framework is ready to be used in our project. Go to the next section for a basic guide on how to get started with actually using the framework.

Using the framework

Just to get things started, we can begin by making a simple HTML paragraph, and then write that to a HTML file. You will want to do the following things first:

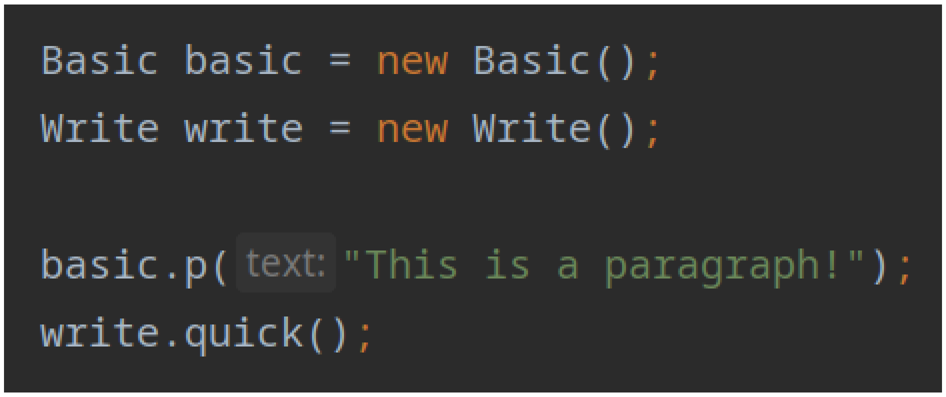
1. Import all the classes from quicktag.html
    1. We only really need the Basic and Write class, but we might as well import the whole package for the sake of simplicity.
2. Make a new Basic object and a new Write object.

After doing these two steps, our code should now look like this:


Main.java

```java
import quicktag.html.*;

public class Main {

    public static void main(String[] args) {

        Basic basic = new Basic();
        Write write = new Write();

    }

}
```

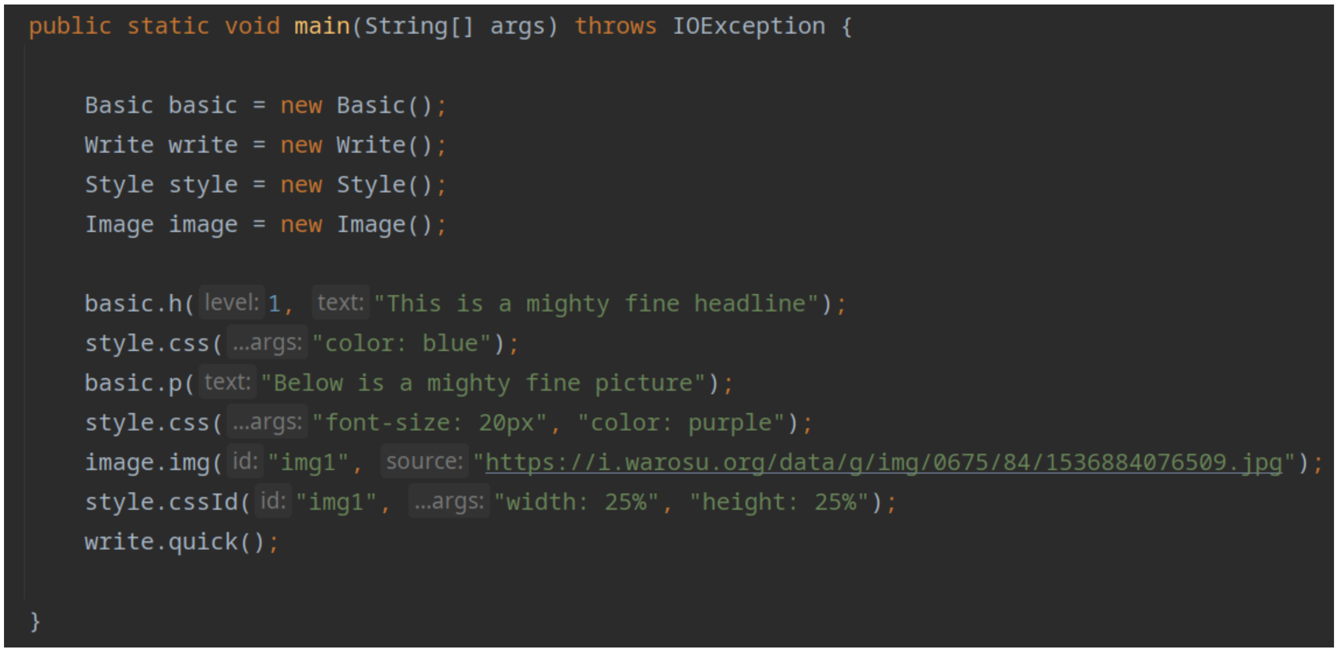
We can now make a paragraph by calling the “p” method from the Basic class, and we then call the “quick” method from the Write class to finish our HTML page. Our code should now look like this:



```java
Basic basic = new Basic();
Write write = new Write();

basic.p( text: "This is a paragraph!");
write.quick();
```

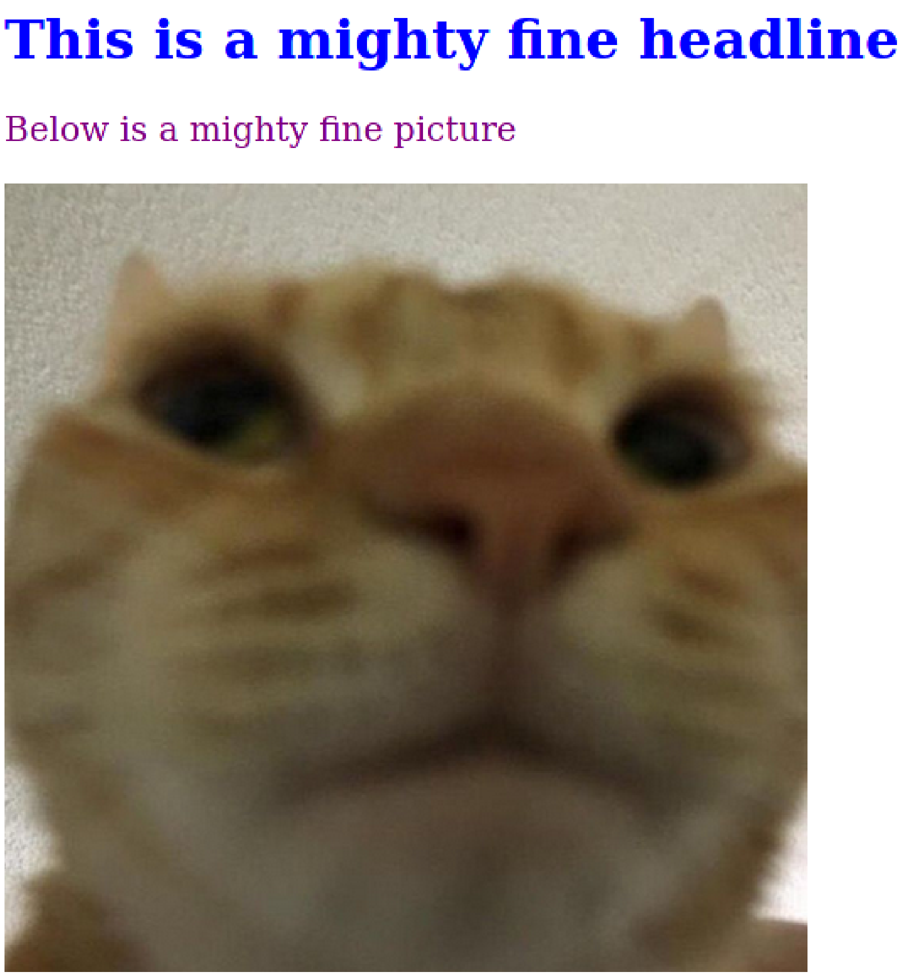
Your IDE may ask you to add a “throws IOException” statement to your main method because of the “quick” method. Now we can run our “Main” program, and it should generate two files: index.html and styles.css.

Once again, we are all done. You are now free to experiment with making more objects, and styling them if you want to. Here is an example that involves both HTML and CSS, so you can get an idea of what is possible:



```
public static void main(String[] args) throws IOException {

    Basic basic = new Basic();
    Write write = new Write();
    Style style = new Style();
    Image image = new Image();

    basic.h( level: 1,  text: "This is a mighty fine headline");
    style.css( ...args: "color: blue");
    basic.p( text: "Below is a mighty fine picture");
    style.css( ...args: "font-size: 20px", "color: purple");
    image.img( id: "img1",  source: "https://i.warosu.org/data/g/img/0675/84/1536884076509.jpg");
    style.cssId( id: "img1",  ...args: "width: 25%", "height: 25%");
    write.quick();

}
```

Open the “index.html” file in a browser, and the page looks like this:



This is a mighty fine headline

Below is a mighty fine picture

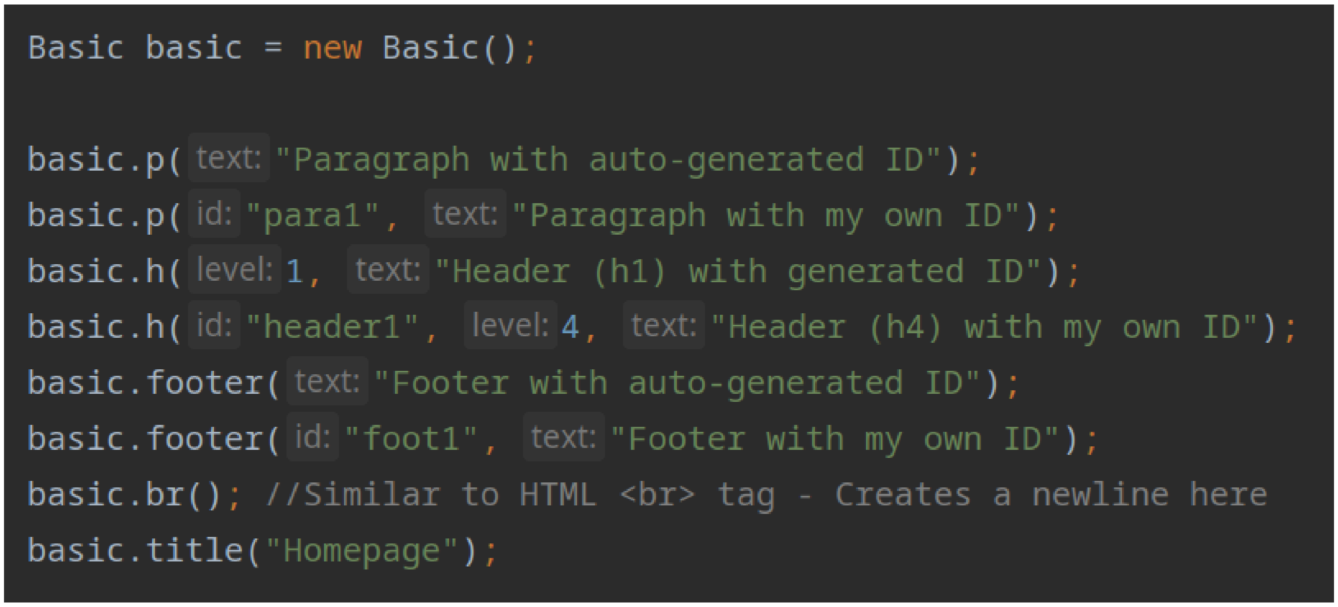
Examples

Basic

The “Basic” class has methods to make some of the most common HTML elements like paragraphs and headers. One unique aspect of the “Basic” class, is that all elements that are not given an ID by the user, will be given a generated ID in the following format: obj1, obj2, obj3 …

This generated ID will be picked up by the basic “CSS” method in the “Style” class, which is why you can style elements without giving them an ID yourself (final example in “How to use the Framework” shows both styling with and without ID). Only methods in “Basic” have this ID generation, as it is intended to be an easy entry point to the framework with methods that don’t require a lot of parameters.

Below is how you can make every single element in “Basic”.

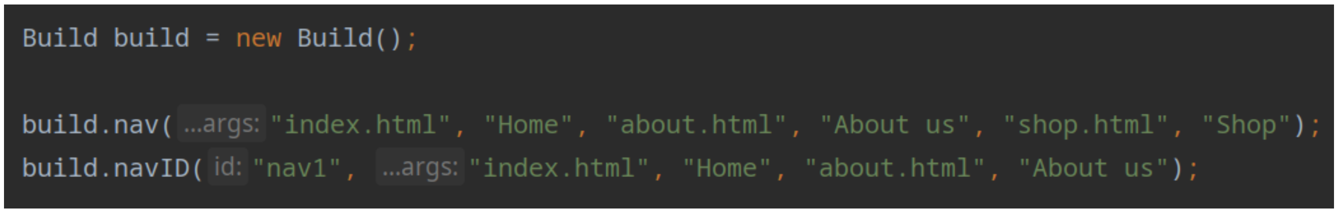


```
Basic basic = new Basic();

basic.p( text: "Paragraph with auto-generated ID");
basic.p( id: "para1",  text: "Paragraph with my own ID");
basic.h( level: 1,  text: "Header (h1) with generated ID");
basic.h( id: "header1",  level: 4,  text: "Header (h4) with my own ID");
basic.footer( text: "Footer with auto-generated ID");
basic.footer( id: "foot1",  text: "Footer with my own ID");
basic.br(); //Similar to HTML <br> tag - Creates a newline here
basic.title("Homepage");
```

Advanced

The more advanced methods are mostly contained in the “Build” class, as these are methods that make things that are not just simple HTML elements. These creations often involve multiple HTML elements as well as CSS, but are still common implementations in many web pages. I will also show an example of the one method in the “Table” class, as it can also be somewhat tricky.

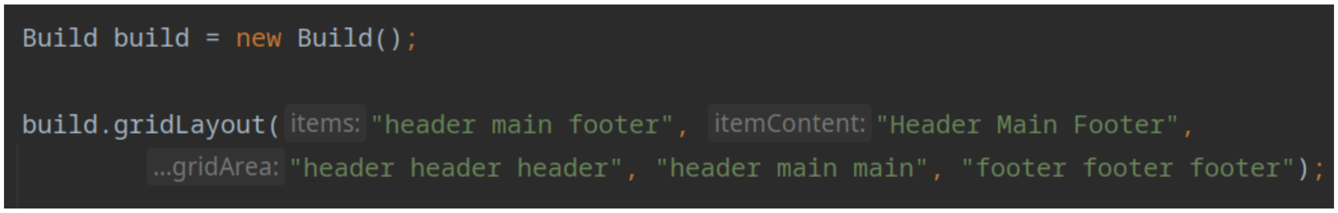
As mentioned in the JavaDoc, the methods for making a navbar, “nav / navID”, require an alternating pattern to their arguments. This means you need to always supply the link to a page first, then the text that you want for that link, like in the below examples:



```
Build build = new Build();

build.nav( ...args: "index.html", "Home", "about.html", "About us", "shop.html", "Shop");
build.navID( id: "nav1", ...args: "index.html", "Home", "about.html", "About us");
```

If we take a look at the first navbar: The link that will say “Home” will take you to index.html, the link that says “About us” will take you to about.html and so on. “navID” works the exact same way, but you can give it an ID of your choice first.

For the gridLayout, you can make one like this:



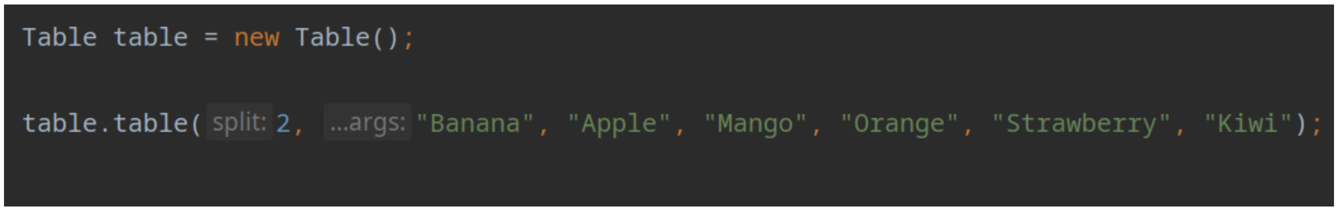
```
Build build = new Build();

build.gridLayout( items: "header main footer", itemContent: "Header Main Footer",
        ...gridArea: "header header header", "header main main", "footer footer footer");
```

You start by defining the items you want in your grid in the first string, and these have to be seperated with a single whitespace (header main footer). The next string is the content inside your grid items, and also require the same seperation by whitespace. Finally for the gridArea, you need to specify the rows at which your grid items will occupy. The above example will make a CSS grid area that looks like this:

‘header header header’
‘header main main’
‘footer footer footer’

So the page is split up in a 3x3 grid, and the above table show which items occupy which cells.

Finally, to make a table, you can do something like this:



```
Table table = new Table();

table.table( split: 2, ...args: "Banana", "Apple", "Mango", "Orange", "Strawberry", "Kiwi");
```

The “split” value, is essentially the amount of columns for your table. So in this case, we would get a table like this:

Banana	Apple
Mango	Orange
Strawberry	Kiwi

Discussion

The Good

The overall highlight for me during this project was during user testing. Almost from the beginning I had hoped to make a framework that was easy to pick up and use, and should require minimum input to get started. To then get feedback from a few users that they liked how easy it was to use, and seeing how fast they were able to finish the tasks I gave them was very rewarding.

I also felt that I was able to get my final client code to be very close to my “dream code” or pseudo code that I wanted. My primary concern had been with what the user had to type in order to make things with my framework. I experimented until I found what I thought was pseudo code as minimalistic as it could be, and tried to make client code like that.

The Bad

Working alone has its own set of challenges, and it’s easy to fall into a sort of echo chamber when you make all the decisions with no opposition from others. That was one reason as to why I found user testing to be so useful. Unless you actively seek out feedback, help or voice chat sessions with teachers it’s easy to get a very narrow focus.

I had an idea very early on to move away from an object based framework, and then focused really intently on one aspect. In doing so I might not have looked into design patterns as much as I could have, and instead just focused on doing my own thing. There were some design patterns that I could have utilized, but by the time I had given them proper attention we were already quite far into the project. Implementing these kinds of patterns would then often require me to restructure the entire project, so I thought it would be better to stick with what I had.

That being said, I could have done a better job at exploring and researching some of these patterns before starting on the actual implementation.

I also would have liked to implement more methods, but found I had to prioritize other things first, and instead stick with covering the more common HTML elements.

Closing Thoughts

All things considered, I had an idea that I thought could be fun to explore and I just went with it. I regret not looking more deeply into some of the theory around design patterns, but quite simply I just enjoyed making the framework, and I hope someone will enjoy messing around with it.