API Specification

Silhouette

Group 12:
Mats Engelien
Lars Erik Faber
Håkon Marthinsen

Contents

1 Introduction **3**
- 1.1 Group Description . . . 3
- 1.2 Delegation of Work . . . 3

2 Background **3**

3 Method **3**
- 3.1 API Design Specification . . . 3
- 3.2 API Design . . . 3
- 3.3 Implementation . . . 3
- 3.4 User Testing . . . 3

4 Design Process/Results **3**
- 4.1 Design Specification . . . 3
- 4.2 Project Structure (File Structure) . . . 4
- 4.3 Client Code . . . 4
- 4.4 User Testing . . . 6
- 4.5 Revised API . . . 6

5 Resulting Framework **6**

6 Discussion **6**

7 Conclusion **6**

8 References **6**

1 Introduction

Link til GitHub: https://github.com/norwen3/Silhouette

1.1 Group Description

1.2 Delegation of Work

Description of Work

Each students writes about scenarios they have contributed with... Workload...

2 Background

Establish existing solutions ...

3 Method

”A section to describe the method under which the framework has been created (can be shared). This section should contain a description for each of the stages given in the ”Framework Design” lecture (API Design Specification, API Design, Implementation and User Testing). You should describe how you have worked, not what you produced, in this section.”

3.1 API Design Specification

3.2 API Design

3.3 Implementation

3.4 User Testing

4 Design Process/Results

4.1 Design Specification

High Level Design Principles...

Design Patterns

Builder Pattern We were very quickly guided towards using the Builder pattern when we first had decided on the Framework-type. The Builder pattern, unlike the Abstract Factory pattern, is a lot more like a ”Fluent interface”, meaning it relies heavily on method cascading. The builder pattern is useful as it attempts to seperate the construction of an object from the way it is represented, this way we can use the same process to create many different looking objects that are fundementally similar. We need to use the builder pattern when we want immutable objects. We want immutable classes because they have a wide range of benefits when we are developing an API. They are simple to use, they are synchronization safe, we only need one constructor, and they are great for maps(which we’re using) to name a few.

The main problem with implementing the builder pattern is the sheer amount of extra code required. The lines will most likely more than double. However, those extra lines of code give us much more readable code and design flexibility. We exchange parameters in the constructor for much prettier method calls.

Design Decisions

Personal Decisions

4.2 Project Structure (File Structure)

Type Reference Documentation

Link to type doc...

4.3 Client Code

Scenarios and Solutions

Scenario 1 Make two rulesets, one that is a regular ruleset and one that is a grid ruleset. Give each ruleset a unique selector. For the regular ruleset, add a blue background and change the text color to #32a852. For the grid, define three columns and two rows of varied size. Lastly, apply both of the rulesets to a Container of type ”header”.

```
RuleSet color = new RuleSet(".color");
color.addRule("background-color", "blue");
color.addRule("color", Color.Hex(#32a852));

Grid grid = new Grid("#grid");
grid.setColumns("1fr", "100px", "2em");
grid.setRows("50%", "120px");

Container myHeader = new Container("header");
myHeader.applySelector(".color");
myHeader.applySelector("#grid");
```

Scenario 2 Make a table whose size changes dynamically, add values to the header row and add values to the rest of the rows as they are generated. Apply a class to the table and set a header color for the table.

```
Table table = new Table();
int col = headerArray.length;
int row = array.length;
RuleSet color = new RuleSet(".tableClass");
color.addRule("background-color", Color.RGB(255,255,255));
ColGroup headerCol = new ColGroup();

table.setSize(row, col);
table.applySelector(".tableClass");

for(int i= 0; i<row; i++){
    for(int j= 0; j<col; j++){
        while(i<1){
            table.insertHead(i,j, headerArray[j]);
        }
        table.insertValue(i,j, array[j])
        if(i<1){
            headerCol.addCol(col, color.toString());
        }
    }
}
```

Scenario 3 Make a frontpage and two article pages and share a common stylesheet between them. Add a simple ruleset that targets the body element and give it a lightgreen background to see that the changes have applied to all the pages. Lastly, generate the code.

```
HTML myWebsite = new HTML("example website");

Page frontPage = new Page("frontpage", "My Front Page");
Page articlePage1 = new Page("article", "My First Article");
page articlePage2 = new Page("article", "My Second Article");

StyleSheet style = new StyleSheet("style.css");

RuleSet bg = new RuleSet("body");
bg.addRule("background-color", "lightgreen");

style.append(bg);

website.link("style.css");
website.addPage(frontPage);
website.addPage(articlePage1);
website.addPage(articlePage2);

website.initialize();
```

Scenario 4 Make a simple form consisting of three input fields for username, password and submit. Label each of the fields accordingly and give the fieldset a legend of ”Login Credentials”.

```
FieldSet myFields = new FieldSet();

Input username = new Input("text");
username.addLabel("Type Username");

Input password = new Input("password");
password.addLabel("Type Password");

Input submit = new Input("submit");

myFields.addFields(username, password, submit);
myFields.addLegend("Login Credentials");
```

Scenario 5 Make a Container of type header and give it a navigation bar with links to all pages previously created. Make the navigation bar into a CSS flexbox so that all anchors be laid horizontally and justify the content to be ”space-around”.

```
Container header = new Container("header");
Container nav = new Container("nav");

Anchor link1 = new AnchorBuilder("Front Page",
    "frontpage.html").build();
Anchor link2 = new AnchorBuilder("First Article",
    "article1.html").build();
Anchor link3 = new AnchorBuilder("Second Article",
    "article1.html").build();

nav.addElements(link1, link2, link3);

FlexBox flexbox = new FlexBox("header nav");
flexbox.setFlexDirection("row");
flexbox.setJustifyContent("space-around");
```

Scenario 6 Make a simple article page about yourself. Write your name in a level 2 heading at the top of the article, then add an unordered list containing containing key points about you. Lastly, add your email in bold text below the text.

Scenario 7 Display a local video on your website and make sure that it supports at least three different file formats.

Scenario 8 Make a questionnaire consisting of two questions. Make it so the first question has two possibilities and one correct answer, and so the second question has three possibilities with multiple correct answers. (Radio and Checkbox)

Scenario 9 Remove underline for all anchors inside nav Containers and change the text color to red on mouse hover for each anchor.

Scenario 10 Give the header a fixed gradient background from blue to red (right).

Scenario 11 Define a grid that serves as a page layout, and give it three columns and one row with equally distributed length. Make sure to wrap them under each other once the screen is narrower than 1000px, such that there are one column and three rows.

4.4 User Testing

Description of setup

The Code

Feedback

4.5 Revised API

5 Resulting Framework

6 Discussion

7 Conclusion

Framvisning av det ferdie API-et

8 References