This section is going to act as a summary to the first four sections of this lesson. We're going to solve a couple of examples together to anchor the main principles of this section into your mind.

Here is the first example:

```css

# header {

```
width: 80%;
margin: 1rem auto;

} ```

First of all, the header is an ID. We dislike IDs in stylesheets and for this reason, I suggest using a class, for example, `.header` will do. The other one is the open-close principle because of the reason that this structure is closed for extension or very much open from modification. Whenever we reference IDs which just tend to stuff everything into the single styling rule that we can. Width and margin is just the beginning, we could do a lot more things in this stylesheet. This is what we would generally discouraged because it will prefer normally adding classes for different responsibilities in terms of styling.

The next example:

`css /* menu styling */ .header ul { /* ... */ } .header ul li { /* ... */ }`

The first mistake is using tag names on the right, it is not efficient at all. The second one is that there is too much nesting going on, so nesting done on three levels, header, `ul`, `li`. It is by far not optimal. It is a lot better if you just created a class for instance for the list items, and just used the styling class consistently.

The third mistake is the violation of a single responsibility principle and the separation of concerns because of the reason that as you can see this unordered list and list item these are styles for a menu. There is even a common indicating that we're dealing with the menu and not with a random unordered list. This way we link the styles of a menu to a header. The header has too many responsibilities. This is how you violated the single responsibility principle.

We could also give a class of the list items, and then we would just have two rules, one of them with the class of the menu, and the other one with the class of the navigation item and menu item. The single responsibility principle, and the open clause principle, are also violated because of another reason.

Whenever we have too much cascading going on it's an obvious sign that we violated the open close principle. When we violate our open/close principle with too much nesting, it makes the rule harder and harder to extend, in this way we violate the separation of concerns as well because whenever we have a new concern, we are not going to extend but we would rather modify our original class.

In this example, you can see two rectangles:

```html

```

```css .content .blogpost-thumbnail { width: 20rem; height: 20rem; font-size: 12px; margin: 1rem; background-color: #ccc; border: 2px black dotted; }

.sidebar .blogpost-thumbnail { width: 10rem; height: 15rem; font-size: 12px; margin: 1rem; background-color: #ccc;
```

border: 2px black dotted;
}

.content, .sidebar { display: inline-block; } ```

These two rectangles are in two different places and their style depends on the container where they are placed in. This is generally a violation of the open clause principle, because obviously it doesn't really make sense to vary the style of a `div` based on its container. For this reason, they're going to create a regular blog post thumbnail and a small blog post thumbnail class, which are extensions of the generic `.blog` post thumbnail class.

```css .blogpost-thumbnail { font-size: 12px; margin: 1rem; background-color: #ccc;
border: 2px black dotted;
}

.blogpost-thumbnail--regular { /* @extend .blogpost-thumbnail */ width: 20rem; height: 20rem; }

.blogpost-thumbnail--small { /* @extend .blogpost-thumbnail */ width: 10rem; height: 15rem;
}

.content, .sidebar { display: inline-block; } ```

There is another problem with this solution though, namely separation of concerns. I would say that dimensions for the blog post thumbnail regular and the blog post thumbnail small classes should normally belong to the containers and not to the thumbnails themselves. The content area and the side bar are both containers, and it's quite certain that they should also have some dimensions.

These dimensions should easily determine the size of the local thumbnail, regular and local thumbnail, small elements, as well. To style these aspects, you're gonna create a thumbnail container and a small thumbnail container class and we're gonna specify all styling information in these classes. This way we symbolize that the content area and the sidebar are both thumbnail containers.

```css .blogpost-thumbnail { font-size: 12px; margin: 1rem; background-color: #ccc;
border: 2px black dotted;
width: 100%; height: 100%; }

.thumbnail-container { width: 20rem; height: 20rem; }

.small-thumbnail-container { width: 10rem; height: 15rem;
}

.content, .sidebar { display: inline-block; } ```

If you argue that this is basically lack of separation of concerns and the content area and the side bar have two responsibilities, first being content area and side bar, and second being thumbnail containers and in this case it makes sense to create an additional div which only acts as a thumbnail container or a small thumbnail container.

Let's see the fourth example. In this example there are quite a few mistakes:

```css

# header .logo-container img {

```
width: 294px !important;
```

} ```

First of all, it's easy to notice that they're using ID. Second, there is too much cascading. That's again, separation of concerns and the open close principle most likely violated. The third one is using tag names on the right. Once more we should just use a proper class name for the name of the image.

It's quite evident to see that with the `!important` rule, this is nothing else but a hack. This is not going to be extensible at all. Therefore we violated the open close principle. There are obviously maintainability concerns as well in general because whenever we apply a hack, there is almost always some untenable solution.
```

Here is the next example:

``` .blogpost-article { color: #27fc4f; font-size: 10px; letter-spacing: 2px; line-height: 1.5; }

.comment-text { color: #27fc4f; font-size: 10px; letter-spacing: 2px; line-height: 1.2;
}

.ad-text { color: #27fc4f; font-size: 10px; letter-spacing: 2px; line-height: 1.4;
} ```

It is evident to conclude that the code itself is WET. We can see the same color, we can see the same font size and some other aspects as well, for instance the letter spacing. In addition it would make sense to introduce a constant for the font color and extract it to a file responsible for the corporate identity. The third concern is that we mix many different aspects in these rules. So we have color, font size, letter spacing, line height, this is a bit too much, we could separate some of these concerns a bit better.

In the sixth example, we have some problems to deal with as well:

`html <section class="patch"></section>`

```css section { /* ... */ }

section.patch { /* ... */ } ```

First of all, `section.patch` is over qualified, it's most likely because we apply this patch as far as well which violates the single responsibility principle. But if we don't violate the single responsibility principle then it's just overqualified or `.patch` is just a bad name.

Let's see the last example:

```html

```

```css .article-content { line-height: 1.5; margin: 1em auto; padding: 1em auto; }

.typography-1 { font-size: 12px; font-family: "Times New Roman"; } .typography-2 { font-size: 14px; font-family: "Arial"; } ```

This is a tricky one because on the surface everything appear to be correct. I mean we could come up with an artificial reason for example we don't have a handle for the `div`s on JavaScript but no one indicated that we need this handles.

So let's not engineer problems where there are no problems in reality. There is one small mistake though. The principle that was violated is separation of concerns because there is coupling between typography and article via the units. Typography one and typography two determines the font size in pixels and all the margins and paddings of the article content are determined by the typography.

On some level, this could be intentional but in practice it could happen that we're better off with using `rem` for margin and padding because then we don't have to worry about what kinds of font views for the purpose of resizing article content.

Even in some CSS that looks very innocent on the surface, these main software engineering principles are violated over and over. Whenever it make sense to correct these mistakes just go ahead, but sometimes it might also happen that we don't want to correct these mistakes because of perfectionism or premature optimization taking too much resources.

It is still worth knowing when we violate some principles of maintainable code, because of the reason that relatively high maintainability is very important for us in the long run. I encourage you to define your own standards and stick to these standards whenever you develop software.
```