The second principle that we're going to discover is the **open/closed principle**.

Software entities can be CSS rules, CSS selectors, CSS modules or even components. For instance, if you recall the blogpost example from the last video, you can figure out that we managed to use inheritance to create a `.normal-blogpost` and the `.featured-blogpost`.

Blogpost itself is open for extension, because if you want to create a premium blogpost for instance, then they could simply derive the `.premium-blogpost` class from the `.blogpost`, and specify the differences in there.

The single responsibility principle (SRP) and the open/closed principle (OCP) often go hand-in-hand. The single responsibility principle is about an object doing one thing at a time. The catch is that according to the open/closed principle, the base object that you would like to extend is closed for modification.

Whenever we customize an object through extending it, all the customization should go to the extended object and not to the base object. The way how the single responsibility principle and the open/closed principle are related, is that whenever we modify the base object so that we can extend it better, then we modify the base object at least most of the time for a secondary reason.

This is why violating the the open/closed principle often induces a violation in the single responsibility principle. This is a very strong concept. The way how you can use this concept in practice is that you have multiple red flags that signal you that you've made a mistake in terms of writing maintainable CSS.

The main benefit of respecting the open/closed principle is the following. Whenever we add new styles that don't change the existing styles, we oftentimes write more maintainable code. This is because a change in an extension or even creating a new extension is most of the time a lot cheaper than changing the base class and figuring out what implication this change has for the whole application or the whole website that we are creating or maintaining.

```css .posts-main { margin: 0 auto; width: 90%; }

.blogpost { width: calc( 33% - 16px ); padding-bottom: 1rem; margin: 0; display: inline-block; }

.blogpost--normal { /* @extend .blogpost; */ background-color: #ccc; border: 8px white solid; }

.blogpost--featured { /* @extend .blogpost; */ background-color: #7f7; border: 8px #050 solid; }

.blogpost__thumbnail { width: 80%; margin: 10% 10% 1rem 10%; }

.blogpost__title { font-size: 1.4rem; margin: 0.25rem 10%; }

.blogpost__summary { margin: 0.25rem 10% 1rem 10%; } ```

As you can see, we are extending the blogpost class by defining a new post type, which is the premium type. All we need to do is extend the original blogpost and then describe the differences in styling in there. For instance, a gold border or a black background, anything can go.

```css .posts-main { margin: 0 auto; width: 90%; }

.blogpost { width: calc( 33% - 16px ); padding-bottom: 1rem; margin: 0; display: inline-block; }

.blogpost--normal { /* @extend .blogpost; */ background-color: #ccc; border: 8px white solid; }

.blogpost--featured { /* @extend .blogpost; */ background-color: #7f7; border: 8px #050 solid; }

.blogpost--premium { /* @extend .blogpost; */ background-color: #111; border: 8px gold solid; color: gold; }

.blogpost__thumbnail { width: 80%; margin: 10% 10% 1rem 10%; }

.blogpost__title { font-size: 1.4rem; margin: 0.25rem 10%; }

.blogpost__summary { margin: 0.25rem 10% 1rem 10%; } ```

And if you wanted to define any other type of blogpost, you could do that easily as well by extending the original blogpost. For this reason, blogpost is open for extension and closed for modification.