Our fourth topic in this lesson is **separation of concerns**. In this section we're going to address what a concern is in CSS, and then we'll look at ways of separating concerns. Then, we're going to get at the big picture of connecting separation of concerns with the single responsibility principle and the open close principle.

A concern is nothing else but a set of CSS features. It can be the box model, it can be typography, colors and backgrounds, or any other sets of CSS features. If you go one level higher, we can also talk about separation of concerns in terms of HTML, CSS, and JavaScript.

Markup normally goes to HTML files. There are some exceptions. For example, React places the markup in our JavaScript file extended with some JSX syntactic sugar. However, separation of concerns still applies, markup should be fully detached from styling. In addition, we have already learned that whenever if create our classes in the markup, we have to separate the classes use for styling and the classes use for functionality. This is also separation of concerns on a different level on class level.

The third way of separating concerns is that different DOM elements would serve different causes. One good example is that oftentimes, we create a container DOM node, a `div` that specifies the spacing and alignment of a certain DOM element. Then we have the DOM element itself that specifies its own look and feel, if you mix these two, it will place two concerns in one DOM note.

It should not only violate separation of concerns but also the single responsibility principle. Separation of concerns can also be discovered down below the CSS hierarchy using different components, modules and libraries. Then we don't separate concerns for example among modules, then one module might start implementing the responsibility of another module.

This is also a violation of the single responsibility principle because of the root cause of not separating concerns properly. Now that we know what separation of concerns is about, let's see how we can separate concerns in practice. Obviously, we're talking about different levels. One level is the CSS class level.

It's very important that, whenever we create rules inside the CSS class, we should not mix multiple CSS features in the same class. Just name the CSS features as different classes, or just use an aggregate class including some mixins. So we've talked about the class selectors there are obviously text and ID's as possible selectors that you can use in style sheets as well.

In general, avoiding HTML tags in stylesheets detaches markup from styling. Markup describes structure and the stylesheets describes styles, this is why there is no place for tags in our CSS, exceptions apply of course. For instance, in the next section we're going to talk about resets and normalizers in CSS, where we reset or normalize all the text inside our application.

For the purpose of re-usability we normally don't use ID attributes. Either because we don't want to create styles for one single element, we prefer creating reusable styles. Recalling the cssguidelin.es example in the video on the open/closed principle we learn that cascading classes is not that optimal. Avoid too much cascading because it creates a rigid structure.

Then we separate concerns properly we would like to link one specific concern to one specific class, and not to a chain of classes inside each other. We already learned in previous sections that it makes perfect sense to separate classes for styling and classes for functionality. Use the notation JS hyphen for the purpose of denoting that the class is intended to be used in JavaScript code.

All the rest of the classes should be used in stylesheets. The reason is that this way a separate styling team and the separate application development team can develop in parallel without any interference. After getting to know all the principles, we can conclude that there is a relationship between separation of concerns, the open/closed principle, and the single responsibility principle.

Once more the single responsibility principle is about our building blocks serving one single responsibility. Separations of concerns is about our building blocks not sharing any responsibilities. Now that we know that our building blocks are one single responsibility and they are fully separated. What we can do using the open/closed principle is that whenever we see some wetness in our building blocks, we abstract them and make them dry.

Let's see an example to figure out what's wrong in this code:

```html

Question 1
Question 2

```

`js $( '#question-header-1' ).css( 'background-color', 'grey' );`

The problem is that the usage of ids are not encourage for this reason they are going to use classes. What we need to do is we have to replace the ID attributes with classes:

```html

Question 1
Question 2

```

`js $( '.js-question-header-1' ) .removeClass( 'question-header' ) .addClass( 'question-header--highlighted' );`

`css .question-header--highlighted { background-color: grey; }`

Let's see the second example:

```css

# main .ratings-table .ratings-table__row td {

```
font-size: 14px;
margin: 0.5em;
color: #444;
```

}```

There are some tag names in CSS on the right position, which is very, very inefficient. And it also violates separation of concerns because of the reason that we have to use a table detail element. The markup is also set in stone. Next, it is not recommended to use IDs in your stylesheets. Next, is too much cascading. Too much cascading is too strict for styles. It's too specific, it's almost never reusable enough. The last problem is with the rows themselves. So you can see that there is a font size setting, there is margin setting and there is color setting. There are different aspects encapsulated in the style of the same element.

Let's move on to the third and final example:

```html

# Title

blabla

```

The first problem here is that the sticky sidebar is most likely made sticky by a JavaScript. This is how you can normally implement some logic that makes the sidebar sticky. Now, in order to access the sticky sidebar in JavaScript, you will need a `.js-sticky-sidebar` handle or something similar. Here we only have a class for styling and that's not enough, that's problem number one.

Another problem is related to the `h2` and the image. Most likely you'll like to style both of these elements, but we are unable to do that without referencing the tag names. Using classes as references instead of tag names are generally better, and for this reason, I'm lacking these class references. That's it!
```