# Multiple Class / ID and Class Selectors

css-tricks.com/multiple-class-id-selectors/

Published February 22, 2010 by Chris Coyier

Can you spot the difference between these two selectors?

```
#header.callout { }
#header .callout { }
```

They look nearly identical, but the top one has no space between "#header" and ".callout" while the bottom one does. This small difference makes a huge difference in what it does. To some of you, that top selector may seem like a mistake, but it's actually a quite useful selector. Let's see the difference, what that top selector means, and exploring more of that style selector.

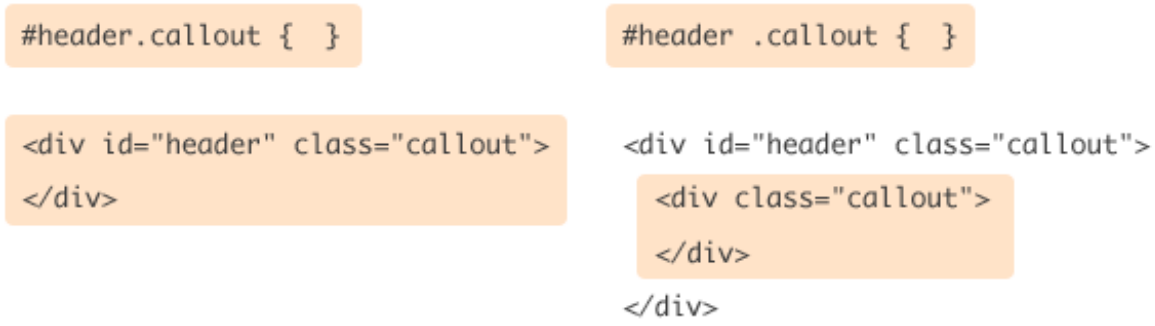
Here is the "plain English" of "#header .callout":

> *Select all elements with the class name callout that are decendents of the element with an ID of header.*

Here is the "plain English" of "#header.callout":

> *Select the element which has an ID of header and also a class name of callout.*

Maybe this graphic will make that more clear:




### Combinations of Classes and IDs

The big point here is that you can target elements that have combinations of classes and IDs by stringing those selectors together without spaces.

### ID and Class Selector

As we covered above, you can target elements by a combination of ID and class.

```
<h1 id="one" class="two">This Should Be Red</h1> #one.two { color: red; }
```

### Double Class Selector

Target an element that has all of multiple classes. Shown below with two classes, but not limited to two.

```
<h1 class="three four">Double Class</h1> .three.four { color: red; }
```

### Multiples

We aren't limited to only two here, we can combine as many classes and IDs into a single selector as we want.

```
.snippet#header.code.red { color: red; }
```

Although bear in mind that's getting a little ridiculous =)

**Example**

So how useful is all this really? Especially with ID's, they are supposed to be unique anyway, so why would you need to combine it with a class? I admit the use cases for the ID versions are slimmer, but there are certainly uses. One of those is overriding styles easily.

```
#header { color: red; }
#header.override { color: black; }
```

The second targets the same element, but overrides the color, instead of having to use:

```
.override { color: black !important }
```

or perhaps prefacing the selector with something even more specific.

More useful is multiple classes and using them in the "object oriented" css style that is all the rage lately. Let's say you had a bunch of divs on a page, and you used multiple various descriptive class names on them:

```
<div class="red border box"></div>
<div class="blue border box"></div>
<div class="green border box"></div>
<div class="red box"></div>
<div class="blue box"></div>
<div class="green box"></div>
<div class="border box"></div>
```

They all share the class "box", which perhaps sets a width or a background texture, something that all of them have in common. Then some of them have color names as classes, this would be for controlling the colors used inside the box. Perhaps green means the box has a greenish background and light green text. A few of them have a class name of "border", presumably these would have a border on them while the rest would not.

So let's set something up:

```
.box { width: 100px; float: left; margin: 0 10px 10px 0; }
.red { color: red; background: pink; }
.blue { color: blue; background: light-blue; }
.green { color: green; background: light-green; }
.border { border: 5px solid black; }
```

Cool, we have a good toolbox going here, where we can create new boxes and we have a variety of options, we can pick a color and if it has a border or not just by applying some fairly semantic classes. Having this class name toolbox also allows us to target unique combinations of these classes. For example, maybe that black border isn't working on the red boxes, let's fix that:

```
.red.border { border-color: #900; }
```

Border color on red box changed because it had both the red class and border class

Based on this demo page.

### Specificity

Also important to note here is that the specificity values of selectors like this will carry the same weight as if they were separate. This is what gives these the overriding power like the example above.

### Browser Compatibility

All good current browsers support this as well as IE back to version 7. IE 6 is rather weird. It selects based on the first selector in the list. So ".red.border" will select based on just ".red", which kinda ruins things. But if you are supporting IE 6, you are used to this kind of tomfoolery anyway and can just fix it with conditional styles.