**93. Creating a Basic Attribute Directive:**

* -: So we already use attribute directives like ngClass and ngStyle.
* It's even easier to understand what to do once we've built our own one.

**Building Custom Directive:**

* So I want to build a directive, which simply highlights a element I hover over, let's say.
* Of course I could achieve the same with normal CSS styles, but this is just a good demo.

1. So I will create a new folder and I will name it basic highlight like this because in there I'll create a new file, basic highlight and now directive.ts

* So before we always had component files.
* Now this is a new element, it's not a component.
* It will be a directive, hence the file name.

b) Now in this file, I'll export a class, which will name BasicHighlightDirective.

c) Again, to be descriptive about what this class is, and to make it a directive, just like with a component where we added at component, here we have to add at directive.

* And directive needs to be imported from at Angular core.
* So make sure to add this import, and we need to pass an object to this decorator to configure this directive.

1. Now, how do we configure a directive? The one thing our directive absolutely needs is a ***selector***.

* Because remember, we do place directives in our template to attach them to elements, so we need to have some way to give Angular that instruction and that is the selector.
* Here, that should also be a unique selector.
* So typically here you use a camel case notation to give it appHighlight as a selector, or here maybe appBasicHighlight like this.
* Now that would select it by element.
* *Now I want to have this attribute style, so I'm going to wrap this in square brackets which means this will now be recognized whenever I add appBasicHighlight without square brackets to an element.*
* Now to show this, we need to do something which gives us some visual glue that this is working.
* So the basic thing is, the most basic use case is, we change, let's say, the background color of the element where we attach this directive.
* For this, we need to get access to the element the directive sits on.
* And the cool thing is, Angular gives us this access.
* We can inject the element the directive sits on into this directive and injection is something we'll take a closer look at in the next course module, in the big next course module, which is about services, I can say that much.
* It basically is an easy way to get access to some other classes without having them to instantiate on our own.

e) We do in check by adding the constructor which every types of class has.

* We don't need to write anything in the constructor body for now.
* But here, on the list of arguments, we list a couple of arguments you want to get whenever an instance of this class here is created.
* And of course Angular is responsible for creating these instances.
* So, therefore, if we tell it to please give us a specific type of argument, this is what injection is.
* Angular will try to create this thing we need and give it to us.
* Well, this thing we need, in this case simply is a reference to the element the directive was placed on.
* So a element reference, this name is totally up to you, but the type is important.

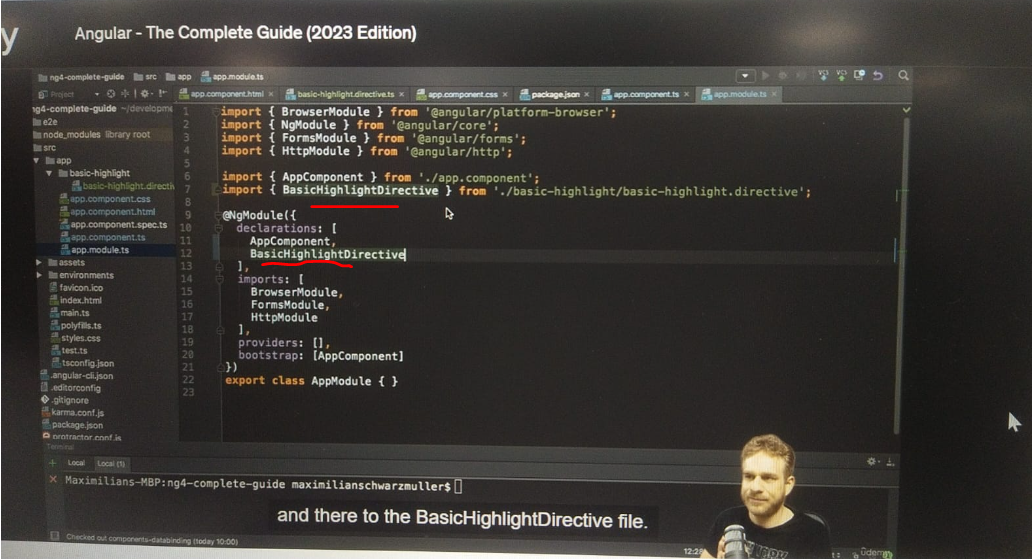
Text

Description automatically generated

* The type has to be ElementRef.
* You might recognize this from at ViewChild.
* There, it also was a reference to some element.
* Here, it is two.
* Now to be able to use this data in our class here, everywhere in the class, we can use a TypeScript shortcut of adding private in front of it, which will make this both a property of this class.
* So property ElementRef and automatically assign this value.
* So this instance, we're getting to this property.
* Now with that, we got access to the element.
* Now we can use it here in our constructor, for example, access to nativeElement and then do something with it.
* Though, a better place than the constructor is OnInIt.
* And just like the, ,uh, component, the directive also has the ngOnInIt lifecycle hook.
* So, here I can therefore add ngOnInIt.
* And in there we could access elementRef, that's the shortcut which automatically gave me this property, access to nativeElement.
* And there access the style, and maybe the background color and set this equal to green so that we can see something changed.
* So what we're doing here is we're getting access to the element, the directive was placed on getting access to that exact element.
* And then we're overriding the style of this element.
* Now let's use this directive

1. **Using the Directive – Specifying in app.module.ts**

* To use it, we have to do two things.
* *First of all, like for a component, we have to inform Angular that we have a new directive.*
* Just like with components, it doesn't scan all our files, so it doesn't know.
* *So we have to go to app module, and here in declarations, we have to add our BasicHighlightDirective and also add the import pointing to the basic highlight folder and there to the BasicHighlightDirective file.*



* Now with this, we can use the directive in our app since we informed Angular.

1. **Using the directive – In HTML file:**

* Now let's use it in our app component HTML file.
* And here below all our lists, I will simply add a new paragraph style me with basic directive, something like that.

Text

Description automatically generated

* And now to this paragraph I can add appBasicHighlight, my own selector.
* We don't need to set any value.
* And importantly we don't use square brackets because, as I already emphasized, the directive name is just a selector we set up here.
* And these square brackets here are not part of the name, it's part of this selector style telling Angular please select it as an attribute on an element.
* And that's just how we add it here, like an attribute of the paragraph.

Text

Description automatically generated

* Now with that, if we save this, we indeed see a green paragraph below our list, because our directive is doing its job here, it's styling it.
* So that's our first basic attribute directive.
* We can enhance this though.