## Inputs & Configuration

**Learning Objectives**

• How to make a directive configurable.

**Configuration**

In the last lecture we finished off our ccCardHover directive. But its not very re-usable; we now want to be able to *configure* it so that it can be used in other situations.

One such configuration parameter is the query selector for the elemenent we want to hide or show, currently it’s hard coded to .card-text, like so:

let part = this.el.nativeElement.querySelector('.card-text');

The first thing to do is move the query selector to a *property* of our directive, but to future-proof ourselves i’m going to set it to a property of an *object*, like so:

config: Object = {

querySelector: '.card-text'

}

This way if we wanted to add further config params in the future we can just add them as properties to our config object.

Next up lets use this config object instead of our hard coded selector. let part = this.el.nativeElement.querySelector(this.config.querySelector); Finally lets make our config property an *input binding* on the directive.

@Input() config: Object = {

querySelector: '.card-text'

}

Now to configure our directive we can add an input property binding on the *same element* the directive, like so:

<

div class="card card-block"

ccCardHover

[config]="{querySelector:'p'}">

①

...

<

div

>

① We’ve configured the querySelector to select on .card-text again, just like before but this time it’s configurable.

But what if we wanted to use our directive like this:

<

div class="card card-block"

[ccCardHover]="{querySelector:'.card-text'}">

①

...

<

div

>

Just like how we’ve seen other built-in directives work.

That’s pretty simple to do, we just need to add an *alias* to the input decorator which matches this decorators *selector*, like so:

@Input('ccCardHover') config: Object = {

querySelector: '.card-text'

}

Now we can use and configure our directive in one statement!

The code for our completed directive looks like this:

@Directive({

selector: "[ccCardHover]"

})

class CardHoverDirective { @HostBinding('class.card-outline-primary') private ishovering: boolean;

@Input('ccCardHover') config: Object = {

querySelector: '.card-text'

};

constructor(private el: ElementRef,

private renderer: Renderer) {

// renderer.setElementStyle(el.nativeElement, 'backgroundColor', 'gray');

}

@HostListener('mouseover') onMouseOver() {

let part = this.el.nativeElement.querySelector(this.config.querySelector);

this.renderer.setElementStyle(part, 'display', 'block');

this.ishovering = true;

}

*script.ts*

import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';

import {

Component,

Directive,

Renderer,

HostListener,

HostBinding,

@HostListener('mouseout') onMouseOut() {

let part = this.el.nativeElement.querySelector(this.config.querySelector);

this.renderer.setElementStyle(part, 'display', 'none');

this.ishovering = false;

}

}

**Summary**

We can configure our directives with standard input property bindings.

To make the syntax look similar to the built-in directives we use an *alias* for the @Input decorator to match the directives selector.

**Listing**

<http://plnkr.co/edit/HMY7tUIztJ79WEmQaj2f?p=preview>

ElementRef,

NgModule,

Input,

Output,

EventEmitter

}

from '@angular/core';

import {BrowserModule} from '@angular/platform-browser';

class Joke {

public setup: string;

public punchline: string;

public hide: boolean;

constructor(setup: string, punchline: string) {

this.setup = setup;

this.punchline = punchline;

this.hide = true;

}

toggle() {

this.hide = !this.hide;

}

}

@Directive({

selector: "[ccCardHover]"

})

class CardHoverDirective {

@HostBinding('class.card-outline-primary') private ishovering: boolean;

@Input('ccCardHover') config: Object = {

querySelector: '.card-text'

};

constructor(private el: ElementRef,

private renderer: Renderer) {

// renderer.setElementStyle(el.nativeElement, 'backgroundColor', 'gray');

}

@HostListener('mouseover') onMouseOver() {

let part = this.el.nativeElement.querySelector(this.config.querySelector);

this.renderer.setElementStyle(part, 'display', 'block');

this.ishovering = true;

}

@HostListener('mouseout') onMouseOut() {

let part = this.el.nativeElement.querySelector(this.config.querySelector);

this.renderer.setElementStyle(part, 'display', 'none');

this.ishovering = false;

}

}

@Component({

selector: 'joke',

template: `

<

div class="card card-block"

[ccCardHover]="{querySelector:'.card-text'}">

<h4 class="card-title">{{data.setup}}</h4>

<p class="card-text"

[style.display]="'none'">{{data.punchline}}</p>

<

/div

>

`

})

class JokeComponent {

@Input('joke') data: Joke;

}

@Component({

selector: 'joke-list',

template: `

<

joke \*ngFor="let j of jokes" [joke]="j"></joke

>

`

})

class JokeListComponent {

jokes: Joke[];

constructor() {

this.jokes = [

new Joke("What did the cheese say when it looked in the mirror?", "Hello-me

(

Halloumi)"),

new Joke("What kind of cheese do you use to disguise a small horse?", "Mask-a-

pony (Mascarpone)"),

new Joke("A kid threw a lump of cheddar at me", "I thought

‘

That

’

s not very

mature

’

"),

];

}

}

@Component({

selector: 'app',

template: `

<

joke-list></joke-list

>

`

})

class AppComponent {

}

@NgModule({

imports: [BrowserModule],

declarations: [

AppComponent,

JokeComponent,

JokeListComponent,

CardHoverDirective

],

bootstrap: [AppComponent]

})

export class AppModule {

}

platformBrowserDynamic().bootstrapModule(AppModule);

## Wrapping Up

In this section we covered how to create a custom directive.

We learned:

* That components *are* directives, but directives with views and templates.
* How to define a directive with the @Directive decorator.
* How to listen to the output events and bind to the input properties of a host element from within a directive.
* How to configure a directive to make it more re-usable.

## Activity

Create a custom *rollover* image directive which shows one image normally and another when the user hovers over it.

The directive will be used like so:

<

img [ccRollover]="

{

'initial':'https://unsplash.it/200/300?image=201',

'over':'https://unsplash.it/200/300?image=202'

}

"/

>

**Steps**

Fork this plunker:

<http://plnkr.co/edit/DcGlVKGc0UDKzqNkPH00?p=preview>

Flesh out the RolloverImageDirective class to implement the functionality you need.

Read any **TODO** comments in the plunker for hints.

**Solution**

When you are ready compare your answer to the solution in this plunker: