ANGULAR 6

Lesson 04

TEMPLATES, STYLES & DIRECTIVES

Objectives

Templates, Styles & Directives

- Template, Styles, View Encapsulation, adding bootstrap to angular app
- Built-in Directives
- Creating Attribute Directive
- Using Renderer to build attribute directive
- Host Listener to listen to Host Events
- Using Host Binding to bind to Host Properties
- Building Structural Directives

TEMPLATES AND STYLE

Template and Styles

View Encapsulation

Adding bootstrap to angular app

DIRECTIVES

Directives

- Directives are instructions to the DOM.
- “Components” are such kind of instructions in the DOM.
- Once we place our selector of our component somewhere in out template, we are instructing angular to add content of our component template and business logic in our typescript code in that place where we use the selector.
- “Components” are directives with templates, But there are also directives without template.
- Two Type of Directives:
  - *Built in directives.*
  - *Custom directives*

Built-in Directives

Angular provides a number of built-in directives, which are attributes we add to our HTML elements that give us dynamic behaviour

- NGIF
  - *The ngIf directive is used when you want to display or hide an element based on a condition.*
  - *The condition is determined by the result of the expression that you pass into the directive.*

```
<div *ngIf="false"></div>     <!-- never displayed -->
<div *ngIf="a > b"></div>     <!-- displayed if a is more than b -->
<div *ngIf="str == 'yes'"></div> <!-- displayed if str is the string "yes"
<div *ngIf="myFunc()"></div>      <!-- displayed if myFunc returns truthy -->
```

Built-in Directives

Capgemini

Angular provides a number of built-in directives, which are attributes we add to our HTML elements that give us dynamic behaviour

- NgSwitch
  - *Sometimes you need to render different elements depending on a given condition. For cases like this, Angular introduces the ngSwitch directive.*

```
<div class="container" [ngSwitch]="myVar">
<div *ngSwitchCase="'A'">Var is A</div>
<div *ngSwitchCase="'B'">Var is B</div>
<div *ngSwitchCase="'C'">Var is C</div>
<div *ngSwitchDefault>Var is something else</div>
</div>
```

Built-in Directives

Capgemini

Angular provides a number of built-in directives, which are attributes we add to our

HTML elements that give us dynamic behaviour

- NgStyle
  - *With the NgStyle directive, you can set a given DOM element CSS properties from Angular expressions.*
  - *The simplest way to use this directive is by doing [style.<cssproperty>]="value". For example:*

```
<div [ngStyle]="{color: 'white', 'background-color': 'blue'}">
Uses fixed white text on blue background
</div>
```

Built-in Directives

- NGCLASS
  - *The NgClass directive, represented by a ngClass attribute in your HTML template, allows you to dynamically set and change the CSS classes for a given DOM element.*
  - *The first way to use this directive is by passing in an object literal. The object is expected to have the keys as the class names and the values should be a truthy/falsy value to indicate whether the class should be applied or not.*

code/built-in-directives/src/styles.css

```
bordered {
        border: 1px dashed black;
        background-color: #eee;
}
```

code/built-in-directives/src/app/ng-class-example/ng-class-example.component.html

```
<div [ngClass]="{bordered: false}">This is never bordered</div>
<div [ngClass]="{bordered: true}">This is always bordered</div>
```

Built-in Directives

- NGCLASS
  - *Alternatively, we can define a classesObj object in our component. And use the object directly:*

```
@Component({
        selector: 'app-ng-class-example',
        templateUrl: './ng-class-example.component.html'
})
export class NgClassExampleComponent implements OnInit {
        isBordered: boolean;
        classesObj: Object;
        classList: string[];
        constructor() {}
        ngOnInit() {
                this.isBordered = true;
                this.classList = ['blue', 'rou
                this.toggleBorder();
        }
        toggleBorder(): void {
                this.isBordered = !this.isBordered;
                this.classesObj = {
                        bordered: this.isBordered
        }; }
```

```
<div [ngClass]="classesObj">
        Using object var. Border {{ classesObj.bordered ? "ON" :
"OFF" }}
</div>
```

Built-in Directives

- NgFor
- The role of this directive is to repeat a given DOM element (or a collection of DOM elements) and pass an element of the array on each iteration.
- The syntax is
  - **ngFor="let item of items"*
  - *The let item syntax specifies a (template) variable that's receiving each element of the items array;*
  - *The items is the collection of items from your controller*

Built-in Directives

- NgFor
  - *Example*

```
In Component
this.cities = ['Miami', 'Sao Paulo', 'New York'];
```

```
In Template
<h4>
Simple list of strings
</h4>

<div *ngFor="let c of cities">
{{ c }}
</div>
```

Built-in Directives

- NgFor
  - *Example*
    - Getting an index
  - *There are times that we need the index of each item when we're iterating an array.*
  - *We can get the index by appending the syntax let idx = index to the value of our ngFor directive, separated by a semi-colon.*
  - *When we do this, ng will assign the current index into the variable we provide (in this case, the variable idx).*
  - *Note that, like JavaScript, the index is always zero based. So the index for first element is 0, 1 for the second and so on.*

```
<div class="ui list" *ngFor="let c of cities; let num = index">
<div class="item">{{ num+1 }} - {{ c }}</div>
</div>
```

Built-in Directives

- NGNONBINDABLE
  - *We use ngNonBindable when we want tell Angular not to compile or bind a particular section of our page.*
  - *Let's say we want to have a div that renders the contents of that content variable and right after we want to point that out by outputting <- this is what {{ content }} rendered next to the actual value of the variable.*

```
<div class='ngNonBindableDemo'>
<span class="bordered">{{ content }}</span>
<span class="pre" ngNonBindable>
&larr; This is what {{ content }} rendered
</span>
</div>
```

Creating Attribute Directive

- We can create directives by annotating a class with the @Directive decorator.

```
    import { Directive } from '@angular/core';
    import { Renderer } from '@angular/core';

    ...
    @Directive({  selector:"[ccCardHover]"})
    class CardHoverDirective {
        constructor(private el: ElementRef, private renderer: Renderer) {
        renderer.setElementStyle(el.nativeElement, 'backgroundColor',
'gray');
    }}
```

```
  <div class="card card-block" ccCardHover>...</div>
```

Host Listener to listen to Host Events

- @HostListener decorator is a function decorator that accepts an event name as an argument. When that event gets fired on the host element it calls the associated function

```
@HostListener('mouseover') onHover() {
    window.alert("hover");
}
```

Host Listener to listen to Host Events

- Lets change our directive to take advantage of the @HostListener

```
   import { HostListener } from '@angular/core’
   ...
   class CardHoverDirective {
      constructor(
            private el: ElementRef,
            private renderer: Renderer) {
   // renderer.setElementStyle(el.nativeElement, 'backgroundColor',
'gray');
   }
   @HostListener('mouseover') onMouseOver() {
      let part = this.el.nativeElement.querySelector('.card-text')
      this.renderer.setElementStyle(part, 'display', 'block');
   }}
```

Using Host Binding to bind to Host Properties

- As well as listening to output events from the host element a directive can also bind to input properties in the host element with @HostBinding.
- This directive can change the properties of the host element, such as the list of classes that are set on the host element as well as a number of other properties.
- Using the @HostBinding decorator a directive can link an internal property to an input property on the host element. So if the internal property changed the input property on the host element would also change.
- We need something, a property on our directive which we can use as a source for binding.

Using Host Binding to bind to Host Properties

```
import { HostBinding } from '@angular/core’
. . .
class CardHoverDirective {
  @HostBinding('class.card-outline-primary') private ishovering: boolean;
  constructor(private el: ElementRef,private renderer: Renderer) {  }
  @HostListener('mouseover') onMouseOver() {
    let part = this.el.nativeElement.querySelector('.card-text');
    this.renderer.setElementStyle(part, 'display', 'block');
    this.ishovering = true;
  }
  @HostListener('mouseout') onMouseOut() {
    let part = this.el.nativeElement.querySelector('.card-text');
    this.renderer.setElementStyle(part, 'display', 'none');
    this.ishovering = false;
  }
}
```

Demo

Demo Custom Directive

Building Structural Directives