4

Directives and elements

Attribute directives, structural directives and built-in Angular elements

Directive

- A directive is a "**modifier**" that can be applicable to HTML elements
- A directive is a class with Metadata
  - Matches a 'marker' in HTML
  - Has directive metadata : **@Directive**
- Types:
  - **Attribute** directives
  - **Structural** directives

Attribute Directives

- Attribute directives change the **appearance or behavior** of an element
- Attribute directives don't modify the DOM structure
- Syntax:

```
<element [directive]="value" > </element>
```

- Build-in attribute directives
  - **ngClass**
  - **ngStyle**

Attribute Directives

ngClass

- Special directive for setting multiple CSS class binding
- Binds to key/value pair
  - Key: name of the class
  - Value: Boolean that indicate whether it should be set

```
<div [ngClass]="{‘article’: true, ‘read’: isRead }">
        …
</div>'
```

Attribute Directives

ngStyle

- Special directive for applying multiple CSS styles
- Binds to key/value pair
  - Key: name of the CSS property
  - Value: value of the CSS propery

```
<div [ngStyle]="{‘color’: ‘red’, ‘background’: ‘grey’ }">
        …
</div>'
```

Implemeting attribute directive

- Use **@Directive**
  - Add a selector (between **[ ]**)
- Inject the **ElementRef** service to access the DOM elements
- Use **Renderer** to make modifications

```
@Directive({ selector: '[hoverImage]' })
export class HoverImageDirective {

 constructor(private el: ElementRef, private renderer: Renderer) {
 }

}
```

Implemeting attribute directive

Get Input data

- Use @Input to get input data (just like components)

```
<img src="images/Bear.jpg" [hoverImage]="hoverUrl" />
```

```
@Directive({ selector: '[hoverImage]' })
export class HoverImageDirective {
 @Input('hoverImage') src string;
 constructor(private el: ElementRef, private renderer: Renderer) {
 }
}
```

Structural Directives

- Structural directives are directive that can **modify** the DOM Structure
  - Add/Remove/repeat elements
- Syntax:

```
<component *directive="value" > <component>
```

- Build-in structural directives
  - ***ngIf**
  - ***ngFor**
  - ***ngSwitch**

Structural Directives

***ngIf**

- Can add/remove HTML elements based on a Boolean
- It will take any expression and turn it into a Boolean
- It does not hide the element, it really **removes the element** from the DOM

```
<table *ngIf="items.length > 0">
  <thead>
    <tr><th>Name</th><th>Price</th></tr>
  </thead>
  <tbody>
          …
   </tbody>
</table>
```

Render the table only if there are items to show

Structural Directives

***ngFor**

- Repeats certain template
  *ngFor="let item of collection"

```
<tbody>
    <tr *ngFor="let item of items">
        <td>{{item.name}}</td>
        <td>{{item.price}}</td>
    </tr>
</tbody>
```

Creates local template variable

Binds to item

Structural Directives

ngSwitch, *ngSwitchCase, *ngSwitchDefault

- Used when displaying **ONE** of multiple items

```
<span [ngSwitch]="color">
  <span *ngSwitchCase=‘red’ »>Red</span>
  <span *ngSwitchCase=’blue’>Blue</span>
  <span *ngSwitchCase=’greeen’ »>Green</span>
  <span *ngSwitchDefault>Black </span>
</span>
```

- ngSwitchDefault is rendered only when none of the switch cases matches.

The Asterisk *

- Why do these structural directives require the * ?
  - Structural directives can manipulate HTML with the help of a template
  - * is a syntactic sugar that hides the template

```
<table *ngIf=”items.length > 0"></table>
```

```
<template [ngIf]=”items.length > 0">
  <table></table>
</template>
```

Implemeting structural directive

- Use @Directive
  - Add a selector (between **[ ]**)
- Inject the **TemplateRef** service to get the embedded template
- Use **ViewContainerRef** container where the template could be attached

Implemeting structural directive

```
@Directive({ selector: '[myUnless]' })
export class UnlessDirective {

  @Input() set myUnless(condition: boolean) {
    if (!condition) {
      this._viewContainer.createEmbeddedView(this._templateRef);
    } else {
      this._viewContainer.clear();
    }
  }
  constructor(
    private _templateRef: TemplateRef,
    private _viewContainer: ViewContainerRef
  ) { }
}
```

```
<p *myUnless="isChecked">hide me!</p>
```

Summary

Attribute directive

- Changes the appearance or behavior
- Built-in: **ngStyle**, **ngClass**
- Uses **ElementRef** and **Renderer** services

Structural directive

- Changes the structure of the DOM
- Built-in: ***ngIf**, ***ngFor, *ngSwitch**
- Uses **TemplateRef** and **ContainerViewRef** services

We cannot use two structural directives on same element

Angular Elements (ng-container)

- A container that allows to group elements
- Doesn't interfere with styles or layout
- Angular doesn't put in the DOM

```
<ng-container *ngIf="store.products">
        <ul *ngFor="let product of store.products">
                <li>{{ product.name }}</li>
        </ul>
</ng-container>
```

Angular Elements (ng-template)

- Define a template (composition of elements) NOT **rendered** by default
- We can give it a name (id) using **(#)**
- Designed to work with structural directives (ngIf for example)

```
<div *ngIf="store.products>0 else NOPRODUCTS ">
        <ul *ngFor="let product of store.products">
                <li>{{ product.name }}</li>
        </ul>
</div>

<ng-template #NOPRODUCTS >
        <p> There are no products </p>
</ng-template>
```

Angular Elements (ng-content)

- Used to project content (view) inside angular components
- We use ng-content tag as a placeholder for dynamic content
  - We call this **Content projection**
- When rendering ng-content is replaced by real content

Child component (child-component)

```
<h1>Title</h1>
<ng-content> </ng-content>
<h6>end<h6>
```

Parent component

```
<child-component>
  <div>
     this is some content
  </div>
</child-component>
```

LAB 4

Displaying list of tasks