Directive In Angular. **Day** 65/100 .

A directive modifies the DOM by changing the appearance, behavior, or layout of DOM elements. Directives just like Component are one of the core building blocks in the Angular framework to build applications. In fact, Angular components are, in large part, directives with templates. In Angular components have assumed a lot of the roles directives used to. In Angular, the major issues related to the template injection and dependency injection are solved with the help of components and issues related to the providing the modification for the generic behavior of the application is done by the help of directives.

So, if we consider the high-level definition of directives, then Directives are acts as a HTML markers on any DOM element (like as an attribute or element name or comment or CSS based style class) which instructed Angular's HTML compiler to attach a specified behaviour on that particular DOM element or to transform that DOM element.

Basic Concept of Directives.

In Angular Framework, one of the most important elements is Directives. And if we analyze the directive in brief, then we will discover that the main building block of the Angular framework which is known as Component is basically a directive. So, in a simple word, each and every Angular component is just a directive with a custom HTML template.

So, in real word when defining a component as the main building block of Angular application, actually we want to say that directives are the main building blocks of Angular applications.
In general, a directive is a TypeScript based function that executes whenever Angular compiler identified it within the DOM element. Directives are used to provide or generate new HTML based syntax which will extend the power of the UI in an Angular Application. Each directive must have a selector name just like the same as a component – either that name can be from Angular predefined patterns like ng–if or a custom developer–defined name which can be any name but need to indicate the main purpose of the directive. Also, every directive can act as an element or an attribute or a class or a comment in the HTML section.

Why Directives Required?

In Angular Framework, Directives always ensure the high–level of reusability of the UI controls throughout the application. With the help of Directives, we can develop UIs with many movable parts and at the same time, we can streamline the development flow for the engineers. The main reason for using directives in any Angular applications are –

1. Reusability – In an Angular application, the directive is a self–sufficient part of the UI. As a developer, we can reuse the directive across the different parts of the application. This is very much useful in any large–scale applications where multiple systems need the same functional elements like search box, date control, etc.

2. Readability – Directive provides much more readability for the developers to understand the production functionality and data flow.
3. Maintainability – One of the main use of directive in any application is the maintainability. We can easily decouple the directive from the application and replace the old one with a new one directives.

@Directive Metadata

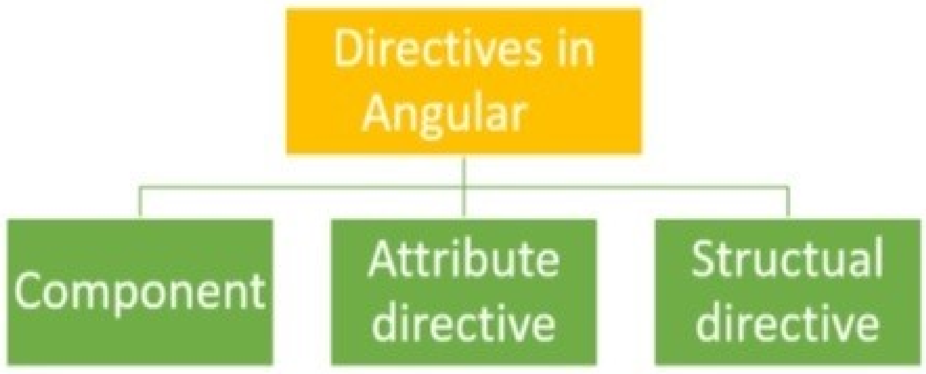
@Directive decorator is used to defining any class as an Angular Directive. We can always define any directive to specify the custom behavior of the elements within the DOM. @Directive() metadata is contained below–mentioned options –

1. selector – The selector property is used to identify the directive within the HTML template. Also, with the help of this property, we can initialize the directive from the DOM elements
2. inputs – It is used to provide the set of data–bound input properties of the directives.
3. Outputs – It is used to enumerates any event properties from the directives.
4. Provider – It is used to inject any provider type like service or components within the directives.
5. ExportAs – It is used to define the name of the directives which can be used to assign a directive as a variable.

Types of Directives

There are three main types of directives in Angular 8:

- Component – Directives with templates.
- Attribute Directives – Directives that change the behavior of a component or element but don't affect the template.
- Structural Directives – Directives that change the behavior of a component or element by affecting the template or the DOM decoration of the UI.



What is an Attribute Directive?

Attribute directives are mainly used for changing the appearance or behavior of a component or a native DOM element. Attribute directives actually modify the appearance or behavior of an element. These directives actually act as a simple HTML attribute for any HTML tag. There some inbuild attribute directive is available in the framework like ngModel. But, we can also create any type of custom attribute–based directive as per our requirement. In that case, use the attribute directive selector name as an attribute within the HTML tag in the HTML code section.

In–Built Attribute Directives

Angular provides some inbuilt Attribute directives which can be used to change the style or attributes of the HTML elements in the DOM. Those attribute directives are −

- ngClass – ngClass directive changes the class attribute that is bound to the component or element it's attached to.
- ngStyle – ngStyle is mainly used to modify or change the element's style attribute. This attribute directive is quite similar to using style metadata in the component class.

What is Structural Directive?

Other types of directives in the Angular framework are the Structural Directive.

Structural directives are mainly used to change the design pattern of the UI DOM elements. In HTML, these directives can be used as a template tag. Using this type of directives, we can change the structure of any DOM elements and can redesign or redecorate those DOM elements. In Angular Framework, there are some system generate structural directives is available like ngIf, ngFor and ngSwitch. We can also create any custom structural directive. The most common example of any custom structural directives is components. Since we can consider every component as a structural directive if that component makes some change in the UI DOM elements style or design. All the system generated structural directives have a template name along with some property value that needs to provide when we define that directives in the HTML code.

In–Build Structural Directive

Angular provides below mentioned built–in directives which can be used within a component to change the elements structure or design.

- ngIf
- ngFor
- ngSwitch

Custom Directive

To create attribute directives, we always need to use or inject the below objects in our custom attribute directive component class. To create an attribute directive, we need to remember the below topics:

1. Import required modules like directives, ElementRef, and renderer from the Angular core library.
2. Create a TypeScript class.
3. Use the @Directive decorator in the class.
4. Set the value of the selector property in the @directive decorator function. The directive will be used, using the selector value on the elements.
5. In the constructor of the class, inject ElementRef and the renderer object.
6. You need to inject ElementRef in the directive's constructor to access the DOM element.
7. You also need to inject the renderer in the directive's constructor to work with the DOM's element style.
8. You need to call the renderer's setElementStyle function. In this function, we need to pass the instance of the current DOM element with the help of ElementRef as a parameter and setting the behavior or property of the current element.

ElementRef – While creating a custom attribute directive, we inject ElementRef in the constructor to access the DOM element. ElementRef provides access to the underlying native element. With the help of ElementRef, we can obtain direct access to the DOM element using its nativeElement property. In that case, ElementRef is behaving just like a service. That's all we need to set the element's color using the browser DOM API.

Renderer– While creating a custom attribute directive, we inject Renderer in the constructor to access the DOM element's style. Actually, we call the renderer's setElementStyle function. In this function, we pass the current DOM element with the help of the ElementRef object and set the required attribute of the current element.

HostListener – Sometimes we may need to access the input property within the attribute directive so that, as per the given attribute directive, we can apply a related attribute within the DOM Element. To trap user actions, we can call different methods to handle user actions. We need to use this method to perform any user action. For that purpose, we need to decorate the method with @HostListener method.

Demo 1: Attribute Directive

In this demo, we will demonstrate how to use inbuilt attribute directives in Angular Framework.

<u>app.component.ts</u>

```
import { Component } from '@angular/core';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls : ['./custom.css']

})

export class AppComponent {

  showColor: boolean = false;


  constructor() { }

  public changeColor(): void {

       this.showColor = !this.showColor;

  }

}
```

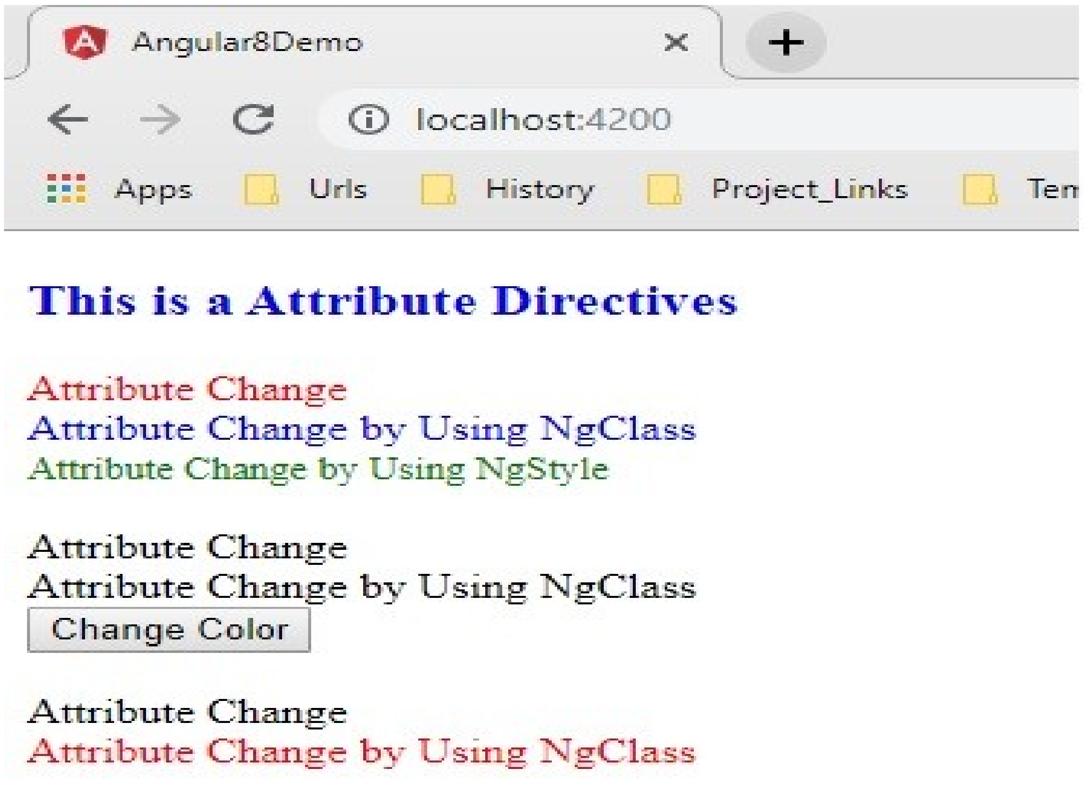
app.component.html

```
<div>
    <h3>This is a Attribute Directives</h3>
    <span [class.red]="true">Attribute Change</span><br />
    <span [ngClass]="{'blue':true}">
        Attribute Change by Using NgClass
    </span><br />
    <span [ngStyle]="{'font-size':'14px','color':'green'}">
         Attribute Change by Using NgStyle
     </span> <br /><br />
    <span [class.cyan]="showColor">Attribute Change</span><br />
    <span [ngClass]="{'brown':showColor}">
          Attribute Change by Using NgClass
    </span><br />
 <input type="button" value="Change Color" (click)="changeColor()" />
    <br /><br />
    <span [class.cyan]="showColor">Attribute Change</span><br />
    <span [ngClass]="{'cyan':showColor, 'red' : !showColor}">
            Attribute Change by Using NgClass
    </span><br />
    <br />
</div>
```

custom.css

```
.red {color:red;}
.blue {color:blue}
.cyan {color : cyan}
.brown {color : brown}
```

Output should be as one shown below:



This is a Attribute Directives

Attribute Change
Attribute Change by Using NgClass
Attribute Change by Using NgStyle

Attribute Change
Attribute Change by Using NgClass
Change Color

Attribute Change
Attribute Change by Using NgClass

Exercise:

1). Create a demo app using ngif directive

2). Create a demo app using ngSwitch directive

3). Create a demo app using Custom Directive – Color Change

Demo2: ngFor

In this demo, we will explain how to use ngFor in our angular application.

app.component.ts

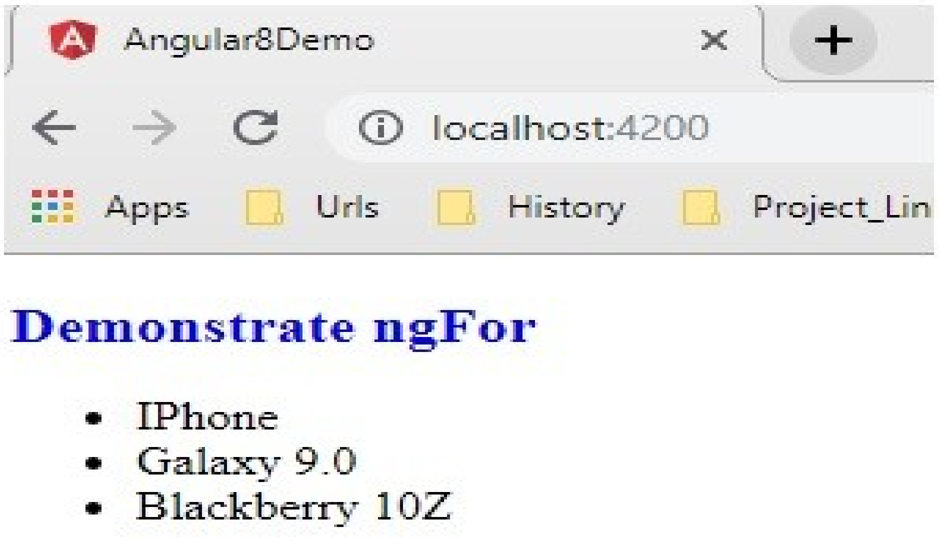
```
import { Component } from '@angular/core';


@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls : ['./custom.css']
})
export class AppComponent {
  productList: Array<string> = ['IPhone','Galaxy
9.0','Blackberry 10Z'];
  constructor() { }
}
```

app.component.html

```
<div>
    <h2>Demonstrate ngFor</h2>
    <ul>
        <li *ngFor="let item of productList">
            {{item}}
        </li>
    </ul>
</div>
```

OutPut:



Demonstrate ngFor

- IPhone
- Galaxy 9.0
- Blackberry 10Z

Best Wishes Lux Tech Academy.