Angular 8 : Online Class

Class – 8

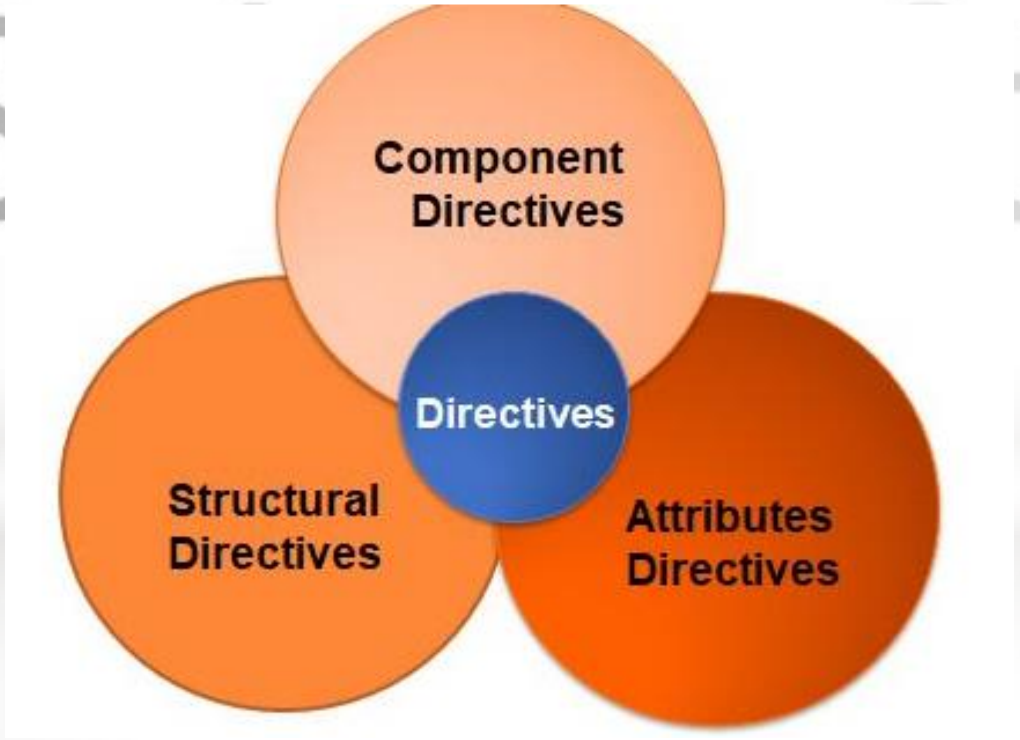
By: Sahosoft Solutions

Presented by : Chandan Kumar

Directive

Basically, directives are used to extend the power of the HTML attributes and to change the appearance or behavior of a DOM element.

Directive in Angular is a javascript class, which is declared as @directive. Angular has 3 types of directives, which are listed below –



Structural directives

Structural directives are a key part of Angular everyone should be familiar with. They are responsible for manipulating DOM through adding, removing or changing the elements. Even if you have never written a structural directive yourself, you have probably been using *ngIf and *ngFor in your templates pretty often. The asterisk (*) states it is a structural directive.

In other words, Structural Directives are directives which change the structure of the DOM by adding or removing its elements. **Structural directives** reconstruct the layout by adding, removing, and replacing elements in **DOM**. The **structural directives** are responsible for shape or reshape the DOM's structure, typically by adding, deleting, or modifying elements. Similar to the other directives, you apply the structural directive to a host element. The directive then performs whatever it is intended to do with that host element. The structural directives are very simple to recognize. An **asterisk (*)** precedes the directive attribute name. It does not require brackets or parentheses like attribute directive. These directives are the way to handle how the component or the element renders in a template.

Structural directives

There are basically 3 built in structural directives available in Angular.

- NgIf (*ngIf)
- NgFor (*ngFor)
- NgSwitch (*ngSwitch)

NgIf

Here we will understand using NgIf with HTML elements. NgIf is an angular directive that is used to add an element subtree for **true** value of expression. NgIf is used as *ngIf="expression". Here if "expression" value is either **false** or **null**, in both cases the element subtree will not be added to DOM.

NgIf with Enum

enum is a value type data type. The enum is used to declare a list of named integer constants. It can be defined using the *enum* keyword directly inside a namespace, class, or structure. The enum is used to give a name to each constant so that the constant integer can be referred using its name.

TypeScript **enum** can also be used with NgIf. To understand this we are creating an **enum** as given below.

numEnum.ts

```
export enum NumEnum
 {
ONE = 1,
TWO = 2,
THREE = 3
}
```

<ng-template> with ngIf

ng-template:

<ng-template> is an angular element for rendering HTML. It is never displayed directly. It can be displayed using structural directive etc. Suppose we have following code in our HTML template.

```
<ng-template>
  <p>Hello World!</p>
</ng-template>
```

When the code runs then the code written inside <ng-template> will not be displayed but there will be a comment.

```
<!---->
```

Before rendering HTML, angular replaces <ng-template> with a comment. <ng-template> can be used with structural directive etc.

NgIf with Else

we will use NgIf with else. In our application else is used when we want to display something for false condition. The else block is used as follows.

```
<div *ngIf="condition; else elseBlock">...</div>
<ng-template #elseBlock>...</ng-template>
```

For else block we need to use <ng-template> element. It is referred by a template reference variable. NgIf will use that template reference variable to display else block when condition is false. The <ng-template> element can be written anywhere in the HTML template but it will be readable if we write it just after host element of NgIf. In the false condition, the host element will be replaced by the body of <ng-template> element.

NgIf-Then-Else

we will use NgIf with then and else. then template is the inline template of NgIf and when it is bound to different value then it displays <ng-template> body having template reference value as then value. NgIfwith then and else is used as follows.

```
<div *ngIf="condition; then thenBlock else elseBlock"></div>
<ng-template #thenBlock>...</ng-template>
<ng-template #elseBlock>...</ng-template>
```

When condition is true, then the <ng-template> with reference variable thenBlock is executed and when condition is false then the <ng-template> with reference variable elseBlock is executed. The value of thenBlock and elseBlockcan be changed at run time. We can have more than one <ng-template> for then and else block and at runtime we can switch to those <ng-template> by changing the value of thenBlock and elseBlock. At one time only one <ng-template> for thenBlock or elseBlock will run.

Expanding *ngIf into template

When we use *ngIf in our code, angular first transform it into template directive and then <template> tag. Find the use of ngIf with asterisk (*)

The asterisk (*) prefix

You noticed the asterisk (*) prefix to the directive name and wondered why it is necessary and what it does.
Here is *ngIf displaying the hero's name if hero exists.

```
<div *ngIf="hero" class="name">{{hero.name}}</div>
```

The asterisk is "syntactic sugar" for something a bit more complicated. Internally, Angular translates the *ngIf attribute into a <ng-template> element, wrapped around the host element, like this.

```
<ng-template [ngIf]="hero">
<div class="name">{{hero.name}}</div>
</ng-template>
```

Expanding *ngIf into template

- The *ngIf directive moved to the <ng-template> element where it became a property binding,[ngIf].
- The rest of the <div>, including its class attribute, moved inside the <ng-template> element.

The first form is not actually rendered, only the finished product ends up in the DOM.

The NgFor and NgSwitch... directives follow the same pattern.

NgIf with Component Elements

Here we will use NgIf with component elements. For the demo we are creating a child component as given below.

msg.component.ts

```
import {Component, Input} from '@angular/core';
@Component({
  selector: 'my-msg',
   template: '<p> Hello {{pname}} </p>
'})
export class MsgComponent
 {
@Input() pname : string;
}
```

Now find the parent HTML template code snippet with component element from person.component.html.

```
<my-msg *ngIf="emp1" [pname] = "emp1.name"> </my-msg>
```

Here we are performing component property binding and using ngIf