<https://www.lynda.com/AngularJS-tutorials/Creating-dashboard-widget/521197/529384-4.html>

Ref : Structural directive in angular2

Directive in angular2

Good blog from angular 2 : <https://blog.thoughtram.io/>

Directive in angular 2 is just as an component. Angular 2 categorizes directives into 3 parts:

1. Directives with templates known as **Components**
2. Directives that creates and destroys DOM elements known as **Structural Directives**
3. Directives that manipulate DOM by changing behavior and appearance known as **Attribute Directives**

Attribute directive : They acts as a attribute to the dom element. They are used to manipulate the DOM.

Structural directive : These type of directive creates , destroyes, recreates the element based on certain conditions.Example : \*ngFor , \*ngIf , ngSwitch

Creating the custom directive :

Directives are created by using the @Directive decorator on a class and specifying a selector. For directives, the selector name must be camelCase and wrapped in square brackets to specify that it is an attribute binding. We're using the @HostListener decorator to listen in on events on the component or element it's attached to.

**Example 1 :**Here in an example below to demonstrate how custom directive actually works. We are applying the mouseover and mouseout event .So, when the mouse is over the element then there will be message gets printed on the console screen. Also the background color gets changes from blue to green.

import {Directive,HostListener,HostBinding} from '@angular/core';  
@Directive({selector : '[mycolor]'})  
export class ColorDirective{  
    private color : string='red';  
    @HostListener('mouseover') mouseover(){  
        console.log('Mouse over');  
        this.color = 'blue';  
    }  
      
    @HostListener('mouseout') mouseout(){  
        console.log('Mouse out')  
        this.color = 'green';  
    }  
      
    @HostBinding('style.color') get init(){          
        return this.color;  
    }  
}

**Example 2 :**Lets create a new directive  with name**myHide**

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

@Directive({selector : '[myHide]'})

export class MyHide{

constructor(el : ElementRef,renderer Renderer){

renderer .setElementStyle(el.nativeElement,'display','none'')

}

}

So , now apply this newely created directive in this way :

**<span myHide>All is well </span>**

Assign 1 : Create a custom directive such that the element should got hidden on mouse over and then reappears on mouseout.

**Example 3 : Create a custom directive so that the element will gets underline when mouse move over it**

import {Directive,HostListener,ElementRef,Renderer} from '@angular/core';  
@Directive({selector : '[myHover]'})  
export class MyHover{  
      
    constructor(private el : ElementRef,private rend : Renderer){  
    }     
    @HostListener('mouseover') mouseOver(){  
        this.change(true)  
    }     
    @HostListener('mouseout') mouseOut(){  
        this.change(false)  
    }     
    change(stt : boolean){  
        if(stt)  
        {  
        this.rend.setElementStyle(this.el.nativeElement,'text-decoration','underline')  
        }  
        else{  
            this.rend.setElementStyle(this.el.nativeElement,'text-decoration','none')  
        }  
    }      
}

**When we apply this : <span myHover>The goon is here</span>**

**Example 4 : Write a directive so that when mouse is hover then text will be of fontWeight bold either it will be normal**

**Hint :**this.render.setElementStyle(this.ele.nativeElement,'fontWeight','bold');

Create Structural Directive :

import {Directive,TemplateRef,ViewContainerRef,Input} from '@angular/core';

@Directive({selector : '[myLoop]'})

export class LoopDirective {

constructor(private templateRef : TemplateRef<any>,private viewContainerRef : ViewContainerRef){

}

@Input('myLoop') set loop(num : number){

for(let i=0;i<num;i++){ this.viewContainerRef.createEmbeddedView(this.templateRef);

}

}

}

We can use like :

**<ul>**

**<li \*myLoop="10">**

**All is good**

**</li>**

**</ul>**

Notes :

TemplateRef and ViewContainerRef

To change DOM layout we should use TemplateRef and ViewContainerRef in our structural directive.

TemplateRef : It represents an embedded template that can be used to instantiate embedded views.

ViewContainerRef: It represents a container where one or more views can be attached.

To use the above classes in our directive, first we need to instantiate them. Instantiate these classes using dependency injection in constructor as following.

constructor (private templateRef: TemplateRef<any>,private viewContainer: ViewContainerRef ) { }

To add host element in DOM layout, we need to call createEmbeddedView() method of ViewContainerRef.

Ex : this.viewContainer.createEmbeddedView(this.templateRef);

If we want to clear view container, call clear() method of ViewContainerRef as given below.

Ex : this.viewContainer.clear();

Now applying the custom conditional directive :

For example :

<div \*myIf=”true”>All is well</div>

The above div will be visible only when we pass **true** to the custom **myIf** directive.

Let’s create the directive :

import {Directive,TemplateRef,ViewContainerRef,Input} from '@angular/core';

@Directive({selector : '[myIf]'})

export class MyCondition{

constructor(private template : TemplateRef<any>,private viewComponent : ViewContainerRef){

}

@Input('myIf') set condition(x : boolean){

if(x){

this.viewComponent.createEmbeddedView(this.template);

}

else{

this.viewComponent.clear();

}

}

}