**Lesson02 what is Directive**

**Notes:-**

**1-A directive allows you to attach a behavior to DOM elements. This behavior could be as simple or as complex as you’d like.\*ngFor and \*ngIf are examples of built-in directives in Angular.**

**2-Directives** in Angular is a **js** class, which is declared as **@directive**. We have 3 directives in Angular. The directives are listed below –

**A-** **Component Directives**

**These form the main class having details of how the component should be processed, instantiated and used at runtime. Ex ng-template , ng-content**

**B-** **Structural Directives**

**A structure directive basically deals with manipulating the DOM elements. Structural directives have a \* sign before the directive. For example, \*ngIf and \*ngFor.**

**C-Attribute Directives**

**Attribute directives deal with changing the look and behavior of the dom element. You can create your own directives as shown below.**

**How to create Directive?**

**1-on the terminal we type the following command**

**ng g directive changeText**

**2-on the app.module.ts we import the directive as following**

import {DateDirective} from './directives/date-directive';

import { StringDirective } from './directives/string.directive';

@NgModule({

declarations: [

//we import the directive on the app.module

StringDirective

]})

**3-on the string-directive.ts we add the following code**

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

@Directive({

selector: '[appChangeText]'})

export class StringDirective {

//we pass the element as parameter on the custom directive

constructor(Element: ElementRef) {

console.log(Element);

//we set each element with inner text

Element.nativeElement.innerText = "Text is changed by changeText Directive. ";}}

**4-on the html page we apply the following code**

<form class="form-horizontal" [formGroup]="empForm" (ngSubmit)="onSubmit()">

<div class="panel panel-primary">

<div class="panel-heading">

<h3 class="panel-title">Create Employee</h3>

</div>

<div class="panel-body">

<div class="form-group">

<label class="col-sm-2 control-label" for="email">Email</label>

<div class="col-sm-8">

//we add the calendar element with specify the calendar Date Format

<p-calendar class="form-control-ng" showIcon="true" utc="true" required="required" formControlName="dateFirstAdvice"

dateFormat="{{ calenderDateFormat }}"></p-calendar>

//we apply custom directive alias name

<span appChangeText>Welcome to</span>

</div></div></div>

<div class="panel-footer">

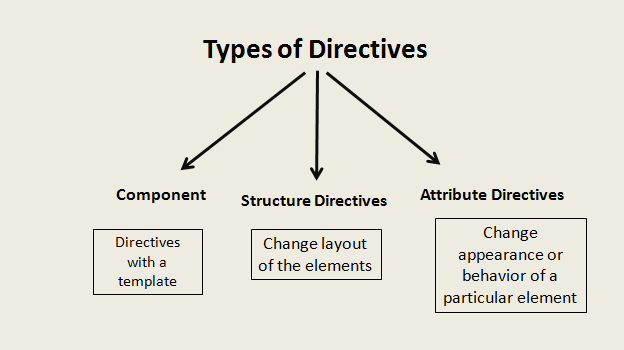
<button class="btn btn-primary" type="submit">Save</button></div>

</div></form>

**Section02 custom Directive**

**Notes:-**

**1- ng-bind, ng-init somewhere in your project. These are the built-in directives that are provided to us by the angular team.**



**There are few steps in the process of creating a custom attribute directive.**

**Step 1. Create a directive by decorating a class with the @Directive decorator.**

**Step 2. Inject Services Element Ref and Renderer**

**Step 3. Register your directive in the module**

**Step 4. Use your directive**

**Notes:-**

**1-When a custom Directive is applied to an HTML element, that element is called the HOST ELEMENT. SO in the above scenario, the custom directive is [app-ch-color] and the HOST ELEMENT will be the div on which you apply [app-ch-color].**

**2-The directive specifies a selector which is what will be looked up in our views. In this case, [myHidden]**

**Steps:-**

**1-on the Directives folder we apply on CMD the following code**

**Ng g directive bgdir**

**2-on the app.module.ts we see that the directive is already added on the declaration section**

import { BgDirective} from './directives/bg-directive.directive';

@NgModule({

declarations: [

BgDirective]});

**3-on the custom directive we add the following code**

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

@Directive({

//this is the key that represent the directive

selector: '[appBgDirective]'})

export class BgDirective {

//we apply ElementRef that represent the element and the Renderer that represent the rendering the DOM element

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

this.changeColor('red');}

changeColor(color: string) {

this.renderer.setElementStyle(this.el.nativeElement, 'color', color);}}

**4-on the html page we call the element with the key**

//The below DIV , we apply the key that apply red color on it

<div appBgDirective> Hello Nepal!!!</div>

<div>Hello Nepal!!! (No Custom Attribute applied)</div>

**5-To capture the event of your Host Element, you need to use @HostListener().**

**import { Directive, ElementRef, Renderer, HostListener } from "@angular/core";**

**@Directive({**

**selector: '[appBgDirective]'})**

**export class BgDirective {**

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

**this.changeColor('red');}**

**//This event called HostListener is called when Click on the element**

**@HostListener('click') foo() {**

**alert('Host Element is clicked');**

**this.changeColor('green');}**

**changeColor(color: string) {**

**//This method will apply red color on the element that we clicked on it**

**this.renderer.setElementStyle(this.el.nativeElement, 'color', color);}}**

**6-If we want to apply design on specific element without needing to call method and apply rendering on it we use @HostBinding**

import { Directive, ElementRef, Renderer, HostListener, HostBinding } from "@angular/core";

@Directive({

selector: '[appBgDirective]'})

export class BgDirective {

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

this.changeColor('red');}

//This means that we assign flag that assign to the property that change //on the same time when we click on the item

@HostBinding('style.border') border : string;

//This event called HostListener is called when Click on the element

@HostListener('click') foo() {

//when we set new value on the DOM element with the border style

this.border = '5px solid green';}

changeColor(color: string) {

this.renderer.setElementStyle(this.el.nativeElement, 'color', color);}}

**Using Multiple @Input() in custom Directives**

**Steps:-**

**1-on the custom Directive we add the following code**

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

@Directive({

selector: '[customTheme]'})

export class BgDirective implements AfterViewInit {

//we define multiple input variable with Input() syntax

@Input() tcolor: string;

@Input() bcolor: string;

@Input() tsize: string;

//we pass Element with it’s own input variables

constructor(private elRef: ElementRef) {}

ngAfterViewInit(): void {

this.tcolor = this.tcolor || 'green';

this.bcolor = this.bcolor || 'cyan';

this.tsize = this.tsize || '20px';

//we can use @HostBinding variable to set default style

this.color= this.tcolor;

this.backgroundColor = this.bcolor;

this.fontSize = this.tsize;}

//we define 2 variable that represent properties of the DOM element

@HostBinding('style.background-color') backgroundColor: string;

@HostBinding('style.font-size') fontSize: string;

//This event called Host Listener is called when Click on the element

@HostListener('click') foo() {

//when we set new value on the DOM element with the border style

this.fontSize = '32px';

this.backgroundColor = 'orange';}

//when the mouse over, the background color change dark grey

@HostListener('mouseover') onMouseOver() {

this.backgroundColor = 'darkgrey';}

//when the mouse leave the background color change to yellow

@HostListener('mouseleave') onMouseLeave() {

this.backgroundColor = 'yellow';}

**2-on the html page we write the following code**

//we set the default color and background color and size

**2-on the html page we add the following code**

<div customTheme> customTheme Directive Demo with Default Settings</div>

//set custom Directive Input variable color and background color and size

<div customTheme tcolor="yellow" bcolor="black" tsize="30px"> customTheme Directive Demo with Custom Settings</div>