**Section04 Understanding Component & Databinding**

**Lesson01 Property And Event Binding**

**Notes:-**

**1-there are three way to communicate between view and code behind and between components**

HTML Elements

Directives

Components

Custom Prosperities and Events

Native Prosperities and Events

Custom Prosperities and Events

**2-by default every property are accessible on the component itself**

**So we will use @Input and @Output which**

**@Input: it will allow child component to be shown on parent component**

**@Output: it will allow pass child component to the parent component**

**Event Emitter is generic type that will used to pass data from child component to parent component**

**3-there are scenario when you have parent component with two child components and you want to send from child 1 and child 2 we have to make @Output with 2 @Input**

**(This is not better approach, because it will make complicated operation)**

**Example:-**

**On the parent component app.component we have two child’s cockpit , server-element**

**<div class="container">**

**<!-- we can use alias name btnCreated as below -->**

**<app cockpit (btnCreated)="onAddServer($event)"**

**(bluePrintCreated)="onAddBluePrint($event)"></app-cockpit>**

**<hr>**

**<div class="row">**

**<div class="col-xs-12">**

**<app-serverelement \*ngFor="let serverElement of serverElements"**

**[srcElement]="serverElement">**

**</app-server-element></div></div></div>**

**//on the code behind**

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

**import { ServerModel } from 'src/app/models/server';**

**@Component({**

**selector: 'app-root',**

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

**styleUrls: ['./app.component.css']})**

**export class AppComponent {**

**serverElements:ServerModel[] = [];**

**onAddServer(serverData:ServerModel){**

**this.serverElements.push({**

**type:'server',**

**name:serverData.name,**

**content:serverData.content});}**

**onAddBluePrint(bluePrintData:ServerModel){**

**this.serverElements.push({**

**type:'blueprint',**

**name:bluePrintData.name,**

**content:bluePrintData.content});}}**

**//the Cockpit. Component we apply @Output Event emitter to pass from child to parent**

**<div class="row">**

**<div class="col-xs-12">**

**<p>Add new servers or blueprints</p>**

**<label>Server Name</label>**

**<input type="text" class="form-control" [(ngModel)]="newServerName">**

**<label>Server Content</label>**

**<input type="text" class="form-control" [(ngModel)]="newServerContent">**

**<br/>**

**<button class="btn btn-primary" (click)="onAddServer()">Add Server</button>**

**<button class="btn btn-primary" (click)="onAddBluePrint()">Add Server BluePrint</button>**

**</div>**

**</div>**

**import { EventEmitter, Output } from '@angular/core';**

**import { Component, OnInit } from '@angular/core';**

**import { ServerModel } from 'src/app/models/server';**

**@Component({**

**selector: 'app-cockpit',**

**templateUrl: './cockpit.component.html',**

**styleUrls: ['./cockpit.component.css']})**

**export class CockpitComponent implements OnInit {**

**constructor() { }**

**ngOnInit(): void {}**

**newServerName = '';**

**newServerContent = '';**

**@Output() serverCreated:EventEmitter<ServerModel> = new EventEmitter();**

**//we can assign alias name for the output event emitter**

**@Output('btnCreated') bluePrintCreated:EventEmitter<ServerModel> = new EventEmitter();**

**//we pass the object from the child to the parent component**

**onAddServer(){**

**var obj:ServerModel ={**

**name:this.newServerName,**

**type:'server',**

**content:this.newServerContent}**

**this.serverCreated.emit(obj);}**

**//we pass the object from the child to the parent component**

**onAddBluePrint(){**

**var obj:ServerModel ={**

**name:this.newServerName,**

**type:'bluePrint',**

**content:this.newServerContent}**

**this.bluePrintCreated.emit(obj);}}**

**//on the server-element component we used @Input to pass from parent to child**

**<div class="panel panel-default">**

**<div class="panel-heading">{{element.name}}</div>**

**<div class="panel-body">**

**<p><strong \*ngIf="element.type === 'server'" style="color:red">{{element.content}}</strong>**

**<em \*ngIf="element.type === 'blueprint'">{{element.content}}</em></p>**

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

**import { Component, Input, OnInit } from '@angular/core';**

**import { ServerModel } from 'src/app/models/server';**

**@Component({**

**selector: 'app-server-element',**

**templateUrl: './server-element.component.html',**

**styleUrls: ['./server-element.component.css']})**

**export class ServerElementComponent implements OnInit {**

**//we can asign alias name and use srcElement instead of element on the parent component**

**@Input('srcElement') element:ServerModel;**

**constructor() { }**

**ngOnInit(): void {}}**

**Lesson02 Understanding Encapsulation**

**Notes:-**

**1-with encapsulation each component has its own CSS file so not overwrite to each other**

**2-shadow copy : its kind of emulates the shadow DOM , the shadow DOM is technology not supported by all browsers where each element has its own DOM styles behind it and you can assign styles as you want to each element**

**(So any style you applied on component it will shadow copy style)**

**3-there are 3 types of encapsulation**

**A-None: will remove shadow copy on the component**

**B-Emulated: the default which applied shadow DOM by default**

**C-Shadow Dom: Use Shadow DOM to encapsulate styles.**

**For the DOM this means using modern Shadow DOM and creating a ShadowRoot for Component's Host Element.**

**Component({**

**selector: 'app-cockpit',**

**templateUrl: './cockpit.component.html',**

**styleUrls: ['./cockpit.component.css'],**

**encapsulation:ViewEncapsulation.ShadowDom})**

**(Be aware that some browser not supported Shadow Dom over emulated because its not feature)**

**Lesson03 using Local References in Templates**

**Notes:-**

**1-local references is very nice feature to get access to some elements in your template and then use that either in your template, but not accessible on your template**

**Example:-**

**<div class="row">**

**<div class="col-xs-12">**

**<p>Add new servers or blueprints</p>**

**<label>Server Name</label>**

**<!-- you can use local reference only on your template not on code behind -->**

**<input type="text" class="form-control" #serverNameInput>**

**<label>Server Content</label>**

**<input type="text" class="form-control" [(ngModel)]="newServerContent">**

**<br/>**

**<button class="btn btn-primary" (click)="onAddServer(serverNameInput)">Add Server</button>**

**<button class="btn btn-primary" (click)="onAddBluePrint(serverNameInput)">Add Server BluePrint</button>**

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

**On the code behind we pass the local reference from view to code behind as below**

**import { EventEmitter, Output, ViewEncapsulation } from '@angular/core';**

**import { Component, OnInit } from '@angular/core';**

**import { ServerModel } from 'src/app/models/server';**

**@Component({**

**selector: 'app-cockpit',**

**templateUrl: './cockpit.component.html',**

**styleUrls: ['./cockpit.component.css']})**

**export class CockpitComponent implements OnInit {**

**constructor() { }**

**ngOnInit(): void {}**

**newServerContent = '';**

**@Output() serverCreated:EventEmitter<ServerModel> = new EventEmitter();**

**@Output('btnCreated') bluePrintCreated:EventEmitter<ServerModel> = new EventEmitter();**

**onAddServer(nameInput){**

**console.log(nameInput.value);**

**var obj:ServerModel ={**

**name:nameInput.value,**

**type:'server',**

**content:this.newServerContent}**

**this.serverCreated.emit(obj);}**

**onAddBluePrint(nameInput){**

**var obj:ServerModel ={**

**name:nameInput.value,**

**type:'bluePrint',**

**content:this.newServerContent}**

**this.bluePrintCreated.emit(obj);}}**

**Lesson04 ViewChild() in Angular 8+**

**Notes:-**

**@ViewChild('serverContentInput', {static: true}) serverContentInput: ElementRef;**

**1-If you DON'T access the selected element in ngOnInit (but anywhere else in your component), set static: false instead!**

**(If you're using Angular 9+, you only need to add {static: true} (if needed) but not {static: false}).**

**2-you should not use this approach to set value on the element on @ViewChild**

**<div class="row">**

**<div class="col-xs-12">**

**<p>Add new servers or blueprints</p>**

**<label>Server Name</label>**

**<input type="text" class="form-control" #serverNameInput>**

**<label>Server Content</label>**

**<input type="text" class="form-control" #serverContentInput>**

**<br/>**

**<button class="btn btn-primary" (click)="onAddServer(serverNameInput)">Add Server</button>**

**<button class="btn btnprimary" (click)="onAddBluePrint(serverNameInput)">**

**Add Server BluePrint</button></div></div>**

**import { ElementRef, EventEmitter, Output, ViewChild, ViewEncapsulation } from '@angular/core';**

**import { Component, OnInit } from '@angular/core';**

**import { ServerModel } from 'src/app/models/server';**

**@Component({**

**selector: 'app-cockpit',**

**templateUrl: './cockpit.component.html',**

**styleUrls: ['./cockpit.component.css']})**

**export class CockpitComponent implements OnInit {**

**constructor() { }**

**ngOnInit(): void {}**

**//we use @ViewChild to gain access to the local reference on code behind**

**@ViewChild('serverContentInput') serverContentInput:ElementRef;**

**@Output() serverCreated:EventEmitter<ServerModel> = new EventEmitter();**

**@Output('btnCreated') bluePrintCreated:EventEmitter<ServerModel> = new EventEmitter();**

**onAddServer(nameInput){**

**console.log(nameInput.value);**

**var obj:ServerModel ={**

**name:nameInput.value,**

**type:'server',**

**content:this.serverContentInput.nativeElement.value}**

**this.serverCreated.emit(obj);}**

**onAddBluePrint(nameInput){**

**var obj:ServerModel ={**

**name:nameInput.value,**

**type:'bluePrint',**

**content:this.serverContentInput.nativeElement.value}**

**this.bluePrintCreated.emit(obj);}}**

**Lesson05 Projection content into component using ng-content**

**Notes:-**

**1-everything you place between the opening and closing tag on your own component is lost by default.**

**2-ng-content is special directive which allow to projected your code which means that you can place code between open and close tag and set <ng-content> on the child component**

**//we set the code we want on the parent component as below**

**<app-server-element \*ngFor="let serverElement of serverElements"  [srcElement]="serverElement">**

**<p>**

**<strong \*ngIf="serverElement.type === 'server'" style="color:red">**

**{{serverElement.content}}**

**</strong>**

**<em \*ngIf="serverElement.type === 'blueprint'">{{serverElement.content}}</em>**

**</p>**

**</app-server-element>**

**On the child component server-element.html we set <ng-content></ng-content> as below**

**<div class="panel panel-default">**

**<div class="panel-heading">**

**{{element.name}}**

**</div>**

**<div class="panel-body">**

**<ng-content></ng-content>**

**</div>**

**</div>**

**Lesson06 Understanding the component lifecycle**

**Notes:-**

**1-life cycle on Angular as below**

**ngOnChanges: called after a bound input property changes , @Input , @Output**

**ngOnInit: called once the component is initialized**

**ngDoCheck: called during every change detection run**

**(Change detection is the system by which angular determines whether something changed on the template of component or inside component)**

**(Like if the @Input variable changed and do re-render the change on the view so it will hook this event as above for each change)**

**ngAfterContentInit: called after content (ng-content) has been projected into view**

**ngAfterContentChecked: called every time the projected content has been checked**

**ngAfterViewInit: called after the components view (and child views) has been initialized**

**ngAfterViewChecked: called every time the view (and child views) has been checked**

**ngOnDestroy: called once the component is about to be destroyed**

**Lesson07 Seeing Life Cycle Hooks in Action**

**Notes:-**

**1-we have to know that with @ViewChild that assigned with local reference it will be accessible value with ngAfterViewInit as below**

**<div class="panel panel-default">**

**<div class="panel-heading">{{element.name}}</div>**

**<div class="panel-heading" #headingEle>{{name}}</div>**

**<div class="panel-body">**

**<!-- it will replaced with the content between caller child component -->**

**<ng-content></ng-content>**

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

**import {AfterContentChecked,AfterContentInit,AfterViewChecked,AfterViewInit,**

**DoCheck,ElementRef,OnDestroy,SimpleChanges,ViewChild} from '@angular/core';**

**import { Component, Input, OnChanges, OnInit } from '@angular/core';**

**import { ServerModel } from 'src/app/models/server';**

**@Component({**

**selector: 'app-server-element',**

**templateUrl: './server-element.component.html',**

**styleUrls: ['./server-element.component.css'],})**

**export class ServerElementComponent implements**

**OnInit,**

**OnChanges,**

**DoCheck,**

**AfterContentInit,**

**AfterContentChecked,**

**AfterViewInit,**

**AfterViewChecked,**

**OnDestroy {**

**//we can asign alias name and use srcElement instead of element on the parent component**

**@Input('srcElement') element: ServerModel;**

**@Input() name: string;**

**@ViewChild('headingEle') heading:ElementRef;**

**//it will called when the instance is created**

**constructor() {console.log('constructor called!');}**

**//it will called when the page initialized**

**ngOnInit(): void {**

**console.log('ngOnInit called');**

**//we see that its not accessable with @ViewChild value**

**console.log(this.heading?.nativeElement?.textContent);}**

**//it will called for any changes for @Input , @Output variables**

**ngOnChanges(changes: SimpleChanges) {**

**console.log('ngOnChanges called');**

**console.log(changes);}**

**//it will called every whenever angular check for any changes there happen ,like event click**

**ngDoCheck(): void {console.log('ngDoCheck Called.');}**

**//it will called after DoCheck and called once when page initialized**

**ngAfterContentInit(): void {console.log('ngAfterContentInit Called.');}**

**//it will called after ngAfterContentInit and**

**ngAfterContentChecked(): void {console.log('ngAfterContentChecked Called.');}**

**//it will called after ngAfterContentChecked and called once when page initialized**

**ngAfterViewInit(): void {**

**console.log('ngAfterViewInit Called.');**

**//we see that its not accessable with @ViewChild value**

**console.log(this.heading.nativeElement.textContent);}**

**//it will called after ngAfterViewInit and called once when page initialized**

**ngAfterViewChecked(): void {console.log('ngAfterViewChecked Called.');}**

**//it will called when destroy view and navigate to new view**

**ngOnDestroy(): void {console.log('ngOnDestroy Called.');}}**

**Lesson08 @ContentChild() in Angular 8+**

**Notes:-**

**1-content child is the same adjustment as for view child apply**

**2-the difference between @ContentChild() and @ViewChild() is that the ContentChild can access with ng-Content with another view while on the @ViewChild is only access on the same view not ng-Content**

**3-@ContentChild() can accessible with ngAfterContentInit() while @ViewChild() can accessible with ngAfterViewInit()**

**<app-serverelement \*ngFor="let serverElement of serverElements"**

**[srcElement]="serverElement" [name]="serverElement.name">**

**<p #paragraph>**

**<strong \*ngIf="serverElement.type === 'server'" style="color:red">**

**{{serverElement.content}}**

**</strong>**

**<em \*ngIf="serverElement.type === 'blueprint'">{{serverElement.content}}</em>**

**</p></app-server-element>**

**//on the code behind**

**//it will called when the page initialized**

**ngOnInit(): void {**

**console.log('ngOnInit called');**

**//we see that its not accessable with @ViewChild value**

**console.log(this.paragraph?.nativeElement?.textContent)}**

**//it will called after DoCheck and called once when page initialized**

**ngAfterContentInit(): void {**

**console.log('ngAfterContentInit Called.');**

**console.log(this.paragraph.nativeElement.textContent)}**