**Interpolation example**  
<button disabled='{{isDisabled}}'>Click Me</button>  
  
**Property binding example**  
<button [disabled]='isDisabled'>Click Me</button>

<img src='{{imagePath}}'/>

<img src='http://www.pragimtech.com/{{imagePath}}' />

<button [disabled]='isDisabled'>Click me</button>

<span [innerHTML]='pageHeader'></span>

With Property binding we enclose the element property name in square brackets  
<button [disabled]='isDisabled'>Click me</button>  
  
We can also use the alternate syntax with bind- prefix. This is known as canonical form  
<button bind-disabled='isDisabled'>Click me</button>

**Some important points to keep in mind when using Property binding**  
  
Remember to enclose the property name with a pair of square brackets. If you omit the brackets, Angular treats the string as a constant and initializes the target property with that string.  
<span [innerHTML]='pageHeader'></span>

**Angular attribute binding**

**Code in employee.component.html file**

<table>

    <thead>

        <tr>

            <th colspan="2">

                Employee Details

            </th>

        </tr>

    </thead>

    <tbody>

        <tr>

            <td>First Name</td>

            <td>{{firstName}}</td>

        </tr>

        <tr>

            <td>Last Name</td>

            <td>{{lastName}}</td>

        </tr>

        <tr>

            <td>Gender</td>

            <td>{{gender}}</td>

        </tr>

        <tr>

            <td>Age</td>

            <td>{{age}}</td>

        </tr>

    </tbody>

</table>

Notice at the moment we have hard-coded **colspan**attribute value to 2 in the HTML. Instead we want to bind to a component class property. So in the employee.component.ts file include 'columnSpan' property as shown below.

export class EmployeeComponent {

    columnSpan: number = 2;

    imagePath: string = 'Tom.png';

    firstName: string = 'Tom';

    lastName: string = 'Hopkins';

    gender: string = 'Male';

    age: number = 20;

}

If we use interpolation to bind columnSpan property of the component class to colspan attribute of the <th> element we get the error - Can't bind to 'colspan' since it isn't a known property of 'th'

<th colspan="{{columnSpan}}">

We get the same error if we use Property Binding

<th [colspan]="columnSpan">

This error is because we do not have a corresponding property in the DOM for colspan attribute. To fix this we have to use attribute-binding in Angular, which sets the colspan attribute. To tell angular framework that we are setting an attribute value we have to prefix the attribute name with attr and a DOT as shown below.

<th [attr.colspan]="columnSpan">

The same is true when using interpolation

<th attr.colspan="{{columnSpan}}">

### Class binding in angular 2

.boldClass{

    font-weight:bold;

}

.italicsClass{

    font-style:italic;

}

.colorClass{

    color:red;

}

In **app.component.ts**, include a button element as shown below. Notice we have set the class attribute of the button element to 'colorClass'. 

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

@Component({

    selector: 'my-app',

    template: `

                <button class='colorClass'>My Button</button>

              `

})

export class AppComponent {

}

**Modify the code in app.component.ts as shown below.**

1. We have introduced a property 'classesToApply' in **AppComponent**class
2. We have also specified class binding for the button element. The word 'class' is in a pair of square brackets and it is binded to the property 'classesToApply'
3. This will replace the existing css classes of the button with classes specified in the class binding

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

@Component({

    selector: 'my-app',

    template: `

                <button class='colorClass' [class]='classesToApply'>My Button</button>

              `

})

export class AppComponent {

    classesToApply: string = 'italicsClass boldClass';

}

Run the application and notice 'colorClass' is removed and these classes (italicsClass & boldClass) are added.

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

@Component({

    selector: 'my-app',

    template: `

                <button class='colorClass' [class.boldClass]='applyBoldClass'>My Button</button>

              `

})

export class AppComponent {

    applyBoldClass: boolean = true;

}

With class binding we can also use ! symbol. Notice in the example below applyBoldClass is set to false. Since we have used ! in the class binding the class is added as expected.

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

@Component({

    selector: 'my-app',

    template: `

                <button class='colorClass' [class.boldClass]='!applyBoldClass'>My Button</button>

              `

})

export class AppComponent {

    applyBoldClass: boolean = false;

}

You can also removed an existing class that is already applied. Consider the following example. Notice we have 3 classes (colorClass, boldClass & italicsClass) added to the button element using the class attribute. The class binding removes the boldClass.

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

@Component({

    selector: 'my-app',

    template: `

                <button class='colorClass boldClass italicsClass'

                        [class.boldClass]='applyBoldClass'>My Button</button>

              `

})

export class AppComponent {

    applyBoldClass: boolean = false;

}

**To add or remove multiple classes use ngClass directive as shown in the example below.**

1. Notice the colorClass is added using the class attribute
2. ngClass is binded to addClasses() method of the AppComponent class
3. addClasses() method returns an object with 2 key/value pairs. The key is a CSS class name. The value can be true or false. True to add the class and false to remove the class.
4. Since both the keys (boldClass & italicsClass) are set to true, both classes will be added to the button element
5. let is a new type of variable declaration in JavaScript.
6. let is similar to var in some respects but allows us to avoid some of the common gotchas that we run into when using var.
7. The differences between let and var are beyond the scope of this video. For our example, var also works fine.
8. As TypeScript is a superset of JavaScript, it supports let

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

@Component({

    selector: 'my-app',

    template: `

                <button class='colorClass' [ngClass]='addClasses()'>My Button</button>

             `

})

export class AppComponent {

    applyBoldClass: boolean = true;

    applyItalicsClass: boolean = true;

    addClasses() {

        let classes = {

            boldClass: this.applyBoldClass,

            italicsClass: this.applyItalicsClass

        };

        return classes;

    }

}

We have included our css classes in a external stylesheet - **styles.css**. Please note we can also include these classes in the styles property instead of a separate stylesheet as shown below.

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

@Component({

    selector: 'my-app',

    template: `

                <button class='colorClass' [ngClass]='addClasses()'>My Button</button>

             `,

    styles: [`

            .boldClass{

                font-weight:bold;

            }

            .italicsClass{

                font-style:italic;

            }

            .colorClass{

                color:red;

            }

             `]

})

export class AppComponent {

    applyBoldClass: boolean = true;

    applyItalicsClass: boolean = true;

    addClasses() {

        let classes = {

            boldClass: this.applyBoldClass,

            italicsClass: this.applyItalicsClass

        };

        return classes;

    }

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Style binding in angular 2

 template: `

                <button style="color:red">My Button</button>

template: `

                <button style='color:red'

                        [style.font-weight]="isBold ? 'bold' : 'normal'">My Button

                </button>

export class AppComponent {

    isBold: boolean = true;

}

 template: `

                <button style='color:red'

                        [style.font-size.px]="fontSize">My Button

                </button>

export class AppComponent {

    fontSize: number = 30;

}

**To set multiple inline styles use NgStyle directive as shown below**

* Notice the color style is added using the style attribute
* ngStyle is binded to **addStyles()** method of the AppComponent class
* **addStyles()** method returns an object with 2 key/value pairs. The key is a style name, and the value is a value for the respective style property or an expression that returns the style value.
* let is a new type of variable declaration in JavaScript.
* let is similar to var in some respects but allows us to avoid some of the common gotchas that we run into when using var.
* The differences between let and var are beyond the scope of this video. For our example, var also works fine.
* As TypeScript is a superset of JavaScript, it supports let

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

@Component({

    selector: 'my-app',

    template: ` <button style='color:red' [ngStyle]="addStyles()">My Button</button>

                    `

})

export class AppComponent {

    isBold: boolean = true;

    fontSize: number = 30;

    isItalic: boolean = true;

    addStyles() {

        let styles = {

            'font-weight': this.isBold ? 'bold' : 'normal',

            'font-style': this.isItalic ? 'italic' : 'normal',

            'font-size.px': this.fontSize

        };

        return styles;

    }

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Angular2 event binding**

A component class property to an HTML element property.

1. Interpolation

2. Property Binding

3. Attribute Binding

4. Class Binding

5. Style Binding

With event binding we can also use the on- prefix alternative as shown below. This is known as the canonical form

<button on-click="onClick()">Click me</button>

Event Binding Example :

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

@Component({

selector: 'my-app',

template: `<button (click)='onClick()' >Click me</button>`

})

export class AppComponent {

onClick(): void {

console.log('Button Clicked');

}

}

Every time we click the button, 'Button Clicked' message is logged to the console. You can see this message under the Console tab, in the browser developer tools.

Event Binding Example :

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

@Component({

selector: 'my-app',

template: `<button (click)='onClick()' >Click me</button>`

})

export class AppComponent {

onClick(): void {

console.log('Button Clicked');

}

}

<button (click)='toggleDetails()'>

{{showDetails ? 'Hide' : 'Show'}} Details

</button>

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

@Component({

selector: 'my-employee',

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

styleUrls: ['app/employee/employee.component.css']

})

export class EmployeeComponent {

columnSpan: number = 2;

firstName: string = 'Tom';

lastName: string = 'Hopkins';

gender: string = 'Male';

age: number = 20;

showDetails: boolean = false;

toggleDetails(): void {

this.showDetails = !this.showDetails;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Two-way data binding in angular 2**

Name : <input [value]='name' (input)='name = $event.target.value'>

<br>

You entered : {{name}}

[value]='name' : This property binding flows data from the component class to element property

(input)='name = $event.target.value' : This event binding flows data in the opposite direction i.e from the element to component class property "name"

$event - Is exposed by angular event binding, and contains the event data. To retrieve the value from the input element use - $event.target.value.

name = $event.target.value - This expression updates the value in the name property in the component class

You entered : {{name}} - This interpolation expression will then display the value on the web page.

<input [value]='name' (input)='name = $event.target.value'>

Like this : Name : <input [(ngModel)]='name'>

Here are the steps to import FormsModule into our AppModule

1. Open app.module.ts file

2. Include the following import statement in it

import { FormsModule } from '@angular/forms';

3. Also, include FormsModule in the 'imports' array of @NgModule

imports: [BrowserModule, FormsModule]

So here is the syntax for using two-way data binding in Angular

<input [(ngModel)]='name'>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Angular ngFor directive**

export class EmployeeListComponent {

employees: any[] = [

{

code: 'emp101', name: 'Tom', gender: 'Male',

annualSalary: 5500, dateOfBirth: '25/6/1988'

},

{

code: 'emp102', name: 'Alex', gender: 'Male',

annualSalary: 5700.95, dateOfBirth: '9/6/1982'

},

{

code: 'emp103', name: 'Mike', gender: 'Male',

annualSalary: 5900, dateOfBirth: '12/8/1979'

},

{

code: 'emp104', name: 'Mary', gender: 'Female',

annualSalary: 6500.826, dateOfBirth: '14/10/1980'

},

];

}

<tr \*ngFor='let employee of employees'>

<td>{{employee.code}}</td>

<td>{{employee.name}}</td>

<td>{{employee.gender}}</td>

<td>{{employee.annualSalary}}</td>

<td>{{employee.dateOfBirth}}</td>

</tr>

<tr \*ngIf="!employees || employees.length==0">

<td colspan="5">

No employees to display

</td>

</tr>

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

import { EmployeeComponent } from './employee/employee.component';

import { EmployeeListComponent } from './employee/employeeList.component';

@NgModule({

imports: [BrowserModule],

declarations: [AppComponent, EmployeeComponent, EmployeeListComponent],

bootstrap: [AppComponent]

})

export class AppModule { }

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

@Component({

selector: 'my-app',

template: `<list-employee></list-employee>`

})

export class AppComponent { }

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Angular ngFor trackBy**

Using trackyBy with ngFor directive :

ngFor directive may perform poorly with large lists

A small change to the list like, adding a new item or removing an existing item may trigger a cascade of DOM manipulations

<table>

<thead>

<tr>

<th>Code</th>

<th>Name</th>

<th>Gender</th>

<th>Annual Salary</th>

<th>Date of Birth</th>

</tr>

</thead>

<tbody>

<tr \*ngFor='let employee of employees'>

<td>{{employee.code}}</td>

<td>{{employee.name}}</td>

<td>{{employee.gender}}</td>

<td>{{employee.annualSalary}}</td>

<td>{{employee.dateOfBirth}}</td>

</tr>

<tr \*ngIf="!employees || employees.length==0">

<td colspan="5">

No employees to display

</td>

</tr>

</tbody>

OR

<tbody>

<tr \*ngFor='let employee of employees; let i=index'> // also this <tr \*ngFor='let employee of employees; let isFirst = first; let isLast = last'>

<td>{{employee.code}}</td>

<td>{{employee.name}}</td>

<td>{{employee.gender}}</td>

<td>{{employee.annualSalary}}</td>

<td>{{employee.dateOfBirth}}</td>

<td>{{i}}</td>

</tr>

<tr \*ngIf="!employees || employees.length==0">

<td colspan="5">

No employees to display

</td>

</tr>

OR

<tr \*ngFor='let employee of employees; let isEven = even; let isOdd = odd'>

<td>{{employee.code}}</td>

<td>{{employee.name}}</td>

<td>{{employee.gender}}</td>

<td>{{employee.annualSalary}}</td>

<td>{{employee.dateOfBirth}}</td>

<td>{{isEven}}</td>

<td>{{isOdd}}</td>

</tr>

<tr \*ngIf="!employees || employees.length==0">

<td colspan="5">

No employees to display

</td>

</tr>

</tbody>

</table>

<br />

<button (click)='getEmployees()'>Refresh Employees</button>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Angular pipes**

* Transform data before display
* Built in pipes include lowercase, uppercase, decimal, date, percent, currency etc
* To apply a pipe on a bound property use the pipe character " | "  
  <td>{{employee.code | uppercase}}</td>
* We can also chain pipes<td>{{employee.dateOfBirth | date:'fullDate' | uppercase }}</td>
* Pass parameters to pipe using colon " : "  
  <td>{{employee.annualSalary | currency:'USD':true:'1.3-3'}}</td>

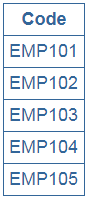
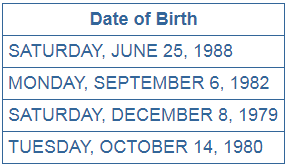
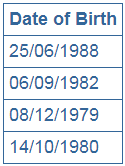
<td>{{employee.dateOfBirth | date:'fullDate'}}</td>

<td>{{employee.dateOfBirth | date:'dd/MM/y'}}</td>

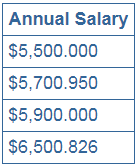
* Custom pipes can be created
* To read more about angular built-in pipes

|  |  |
| --- | --- |
| **Pipe** | **URL** |
| Date | <https://angular.io/api/common/DatePipe> |
| Decimal | <https://angular.io/api/common/DecimalPipe> |
| Currency | <https://angular.io/api/common/CurrencyPipe> |
| Percent | <https://angular.io/api/common/PercentPipe> |

**Please note:** If you get the following error, chances are that your date is not in mm/dd/yyyy format. To fix this error please change the date format to mm/dd/yyyy or create a custom pipe

<td>{{employee.code | uppercase}}</td>  
  
**Output :**  
  
  
In this example, we have chained date and uppercase pipes.  
<td>{{employee.dateOfBirth | date:'fullDate' | uppercase }}</td>  
  
**Output :**  
  
  
In this example we are passing a single parameter to date pipe. With the parameter we specified we want the date format to be dd/mm/yyyy  
<td>{{employee.dateOfBirth | date:'dd/MM/y'}}</td>  
  
Output :   
  
  
For the list of date pipe parameter values please check the following article  
<https://angular.io/api/common/DatePipe>  
  
In this example we are passing 3 parameters to the currency pipe  
<td>{{employee.annualSalary | currency:'USD':true:'1.3-3'}}</td>

1. The first parameter is the currencyCode
2. The second parameter is boolean - True to display currency symbol, false to display currency code
3. The third parameter ('1.3-3') specifies the number of integer and fractional digits

Output :   


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Angular custom pipe

**Step 1 :** To achieve this let's create a custom pipe called employeeTitlePipe. Right click on the "employee" folder and add a new TypeScript file. Name it "employeeTitle.pipe.ts". Copy and paste the following code.  
  
**Code Explanation :**

* Import Pipe decorator and PipeTransform interface from Angular core
* Notice "EmployeeTitlePipe" class is decorated with Pipe decorator to make it an Angular pipe
* name property of the pipe decorator is set to employeeTitle. This name can then be used on any HTML page where you want this pipe functionality.
* EmployeeTitlePipe class implements the PipeTransform interface. This interface has one method transform() which needs to be implemented.
* Notice the transform method has 2 parameters. value parameter will receive the name of the employee and gender parameter receives the gender of the employee. The method returns a string i.e Mr. or Miss. prefixed to the name of the employee depending on their gender.

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

    name: 'employeeTitle'

})

export class EmployeeTitlePipe implements PipeTransform {

    transform(value: string, gender: string): string {

        if (gender.toLowerCase() == "male")

            return "Mr." + value;

        else

            return "Miss." + value;

    }

}

**Step 2 :** Register "EmployeeTitlePipe" in the angular module where we need it. In our case we need it in the root module. So in app.module.ts file, import the EmployeeTitlePipe and include it in the "declarations" array of NgModule decorator

import { EmployeeTitlePipe } from './employee/employeeTitle.pipe'

@NgModule({

    imports: [BrowserModule],

    declarations: [AppComponent, EmployeeComponent,

                   EmployeeListComponent, EmployeeTitlePipe],

    bootstrap: [AppComponent]

})

export class AppModule { }

**Step 3 :** In "employeeList.component.html" use the "EmployeeTitlePipe" as shown below. Notice we are passing employee gender as an argument for the gender parameter of our custom pipe. Employee name gets passed automatically.

<tr \*ngFor='let employee of employees;'>

    <td>{{employee.code}}</td>

    <td>{{employee.name | employeeTitle:employee.gender}}</td>

    <td>{{employee.gender}}</td>

    <td>{{employee.annualSalary}}</td>

    <td>{{employee.dateOfBirth}}</td>

</tr>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Angular 2 container and nested components

**employeeCount.component.ts**

* We have set select='employee-count'. We can use this selector as a directive where we want to use this component. We are going to nest this component inside EmployeeListComponent using employee-count selector as a directive.
* We have 3 properties (all, male and Female). At the moment we have hard coded the values. In our next video we will discuss how to pass the values for these properties from the container component i.e from the EmployeeListComponent.

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

@Component({

    selector: 'employee-count',

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

    styleUrls: ['app/employee/employeeCount.component.css']

})

export class EmployeeCountComponent {

    all: number = 10;

    male: number = 5;

    female: number = 5;

}

Add a new StyleSheet to the employee folder. Name it employeeCount.component.css. Copy and paste the following style class.  
  
**employeeCount.component.css**

.radioClass {

    color: #369;

    font-family: Arial, Helvetica, sans-serif;

    font-size: large;

}

Now let's add the view template for EmployeeCountComponent. Add a new HTML page to the employee folder. Name it employeeCount.component.html. Copy and paste the following HTML.  
  
**employeeCount.component.html**  
Notice we have 3 radio buttons and bound them to the 3 properties (all, male, female) we have in the component class. We are using interpolation for data-binding.

<span class="radioClass">Show : </span>

<input type="radio" name="options" />

<span class="radioClass">{{"All(" + all + ")"}}</span>

<input name="options" type="radio">

<span class="radioClass">{{"Male(" + male + ")"}}</span>

<input name="options" type="radio">

<span class="radioClass">{{"Female(" + female + ")"}}</span>

Nest EmployeeCountComponent in EmployeeListComponent component. To do this, use EmployeeCountComponent selector (employee-count) as a directive <employee-count></employee-count> on EmployeeListComponent component as shown below.

<employee-count></employee-count>

<employee-count [all]="getTotalEmployeesCount()"

                [male]="getMaleEmployeesCount()"

                [female]="getFemaleEmployeesCount()">

</employee-count>

<br /><br />

<table>

    <thead>

        <tr>

            <th>Code</th>

            <th>Name</th>

            <th>Gender</th>

            <th>Annual Salary</th>

            <th>Date of Birth</th>

        </tr>

    </thead>

    <tbody>

        <tr \*ngFor="let employee of employees;">

            <td>{{employee.code | uppercase}}</td>

            <td>{{employee.name | employeeTitle:employee.gender }}</td>

            <td>{{employee.gender}}</td>

            <td>{{employee.annualSalary | currency:'USD':true:'1.3-3'}}</td>

            <td>{{employee.dateOfBirth | date:'dd/MM/y'}}</td>

        </tr>

        <tr \*ngIf="!employees || employees.length==0">

            <td colspan="5">

                No employees to display

            </td>

        </tr>

    </tbody>

</table>

Finally, declare EmployeeCountComponent in the module.ts file. Please make sure you import the component first and then add it to the declarations array of the @NgModule decorator.

import { EmployeeCountComponent } from './employee/employeeCount.component';

@NgModule({

    imports: [BrowserModule, FormsModule],

    declarations: [AppComponent, EmployeeComponent,

        EmployeeListComponent, EmployeeTitlePipe, EmployeeCountComponent],

    bootstrap: [AppComponent]

})

export class AppModule { }

At this point, run the project and you should see employee count radio buttons and the employee list.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Angular component input properties

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

@Component({

    selector: 'employee-count',

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

    styleUrls: ['app/employee/employeeCount.component.css']

})

export class EmployeeCountComponent {

    @Input()

    all: number;

    @Input()

    male: number;

    @Input()

    female: number;

}

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

@Component({

    selector: 'list-employee',

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

    styleUrls: ['app/employee/employeeList.component.css']

})

export class EmployeeListComponent {

    employees: any[];

    constructor() {

        this.employees = [

            {

                code: 'emp101', name: 'Tom', gender: 'Male',

                annualSalary: 5500, dateOfBirth: '6/25/1988'

            },

            {

                code: 'emp102', name: 'Alex', gender: 'Male',

                annualSalary: 5700.95, dateOfBirth: '9/6/1982'

            },

            {

                code: 'emp103', name: 'Mike', gender: 'Male',

                annualSalary: 5900, dateOfBirth: '12/8/1979'

            },

            {

                code: 'emp104', name: 'Mary', gender: 'Female',

                annualSalary: 6500.826, dateOfBirth: '10/14/1980'

            },

            {

                code: 'emp105', name: 'Nancy', gender: 'Female',

                annualSalary: 6700.826, dateOfBirth: '12/15/1982'

            },

        ];

    }

    getTotalEmployeesCount(): number {

        return this.employees.length;

    }

    getMaleEmployeesCount(): number {

        return this.employees.filter(e => e.gender === 'Male').length;

    }

    getFemaleEmployeesCount(): number {

        return this.employees.filter(e => e.gender === 'Female').length;

    }

}

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

@Component({

    selector: 'list-employee',

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

    styleUrls: ['app/employee/employeeList.component.css']

})

export class EmployeeListComponent {

    employees: any[];

    constructor() {

        this.employees = [

            {

                code: 'emp101', name: 'Tom', gender: 'Male',

                annualSalary: 5500, dateOfBirth: '6/25/1988'

            },

            {

                code: 'emp102', name: 'Alex', gender: 'Male',

                annualSalary: 5700.95, dateOfBirth: '9/6/1982'

            },

            {

                code: 'emp103', name: 'Mike', gender: 'Male',

                annualSalary: 5900, dateOfBirth: '12/8/1979'

            },

            {

                code: 'emp104', name: 'Mary', gender: 'Female',

                annualSalary: 6500.826, dateOfBirth: '10/14/1980'

            },

            {

                code: 'emp105', name: 'Nancy', gender: 'Female',

                annualSalary: 6700.826, dateOfBirth: '12/15/1982'

            },

        ];

    }

    getTotalEmployeesCount(): number {

        return this.employees.length;

    }

    getMaleEmployeesCount(): number {

        return this.employees.filter(e => e.gender === 'Male').length;

    }

    getFemaleEmployeesCount(): number {

        return this.employees.filter(e => e.gender === 'Female').length;

    }

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Angular component output properties

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

export class EmployeeCountComponent {

    @Input()

    all: number;

    @Input()

    male: number;

    @Input()

    female: number;

    // Holds the selected value of the radio button

    selectedRadioButtonValue: string = 'All';

    // The Output decorator makes the property an Output property

    // EventEmitter class is used to create the custom event

    // When the radio button selection changes, the selected

    // radio button value which is a string gets passed to the

    // event handler method. Hence, the event payload is string.

    @Output()

    countRadioButtonSelectionChanged: EventEmitter<string> =

                                        new EventEmitter<string>();

    // This method raises the custom event. We will bind this

    // method to the change event of all the 3 radio buttons

    onRadioButtonSelectionChange() {

        this.countRadioButtonSelectionChanged

            .emit(this.selectedRadioButtonValue);

    }

<span class="radioClass">Show : </span>

<input name='options' type='radio' value="All"

       [(ngModel)]="selectedRadioButtonValue"

       (change)="onRadioButtonSelectionChange()">

<span class="radioClass">{{'All(' + all + ')'}}</span>

<input name="options" type="radio" value="Male"

       [(ngModel)]="selectedRadioButtonValue"

       (change)="onRadioButtonSelectionChange()">

<span class="radioClass">{{"Male(" + male + ")"}}</span>

<input name="options" type="radio" value="Female"

       [(ngModel)]="selectedRadioButtonValue"

       (change)="onRadioButtonSelectionChange()">

<span class="radioClass">{{"Female(" + female + ")"}}</span>

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

@Component({

    selector: 'list-employee',

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

    styleUrls: ['app/employee/employeeList.component.css']

})

export class EmployeeListComponent {

    employees: any[];

    // This property keeps track of which radio button is selected

    // We have set the default value to All, so all the employees

    // are displayed in the table by default

    selectedEmployeeCountRadioButton: string = 'All';

    constructor() {

        this.employees = [

            {

                code: 'emp101', name: 'Tom', gender: 'Male',

                annualSalary: 5500, dateOfBirth: '6/25/1988'

            },

            {

                code: 'emp102', name: 'Alex', gender: 'Male',

                annualSalary: 5700.95, dateOfBirth: '9/6/1982'

            },

            {

                code: 'emp103', name: 'Mike', gender: 'Male',

                annualSalary: 5900, dateOfBirth: '12/8/1979'

            },

            {

                code: 'emp104', name: 'Mary', gender: 'Female',

                annualSalary: 6500.826, dateOfBirth: '10/14/1980'

            },

            {

                code: 'emp105', name: 'Nancy', gender: 'Female',

                annualSalary: 6700.826, dateOfBirth: '12/15/1982'

            },

            {

                code: 'emp106', name: 'Steve', gender: 'Male',

                annualSalary: 7700.481, dateOfBirth: '11/18/1979'

            },

        ];

    }

    getTotalEmployeesCount(): number {

        return this.employees.length;

    }

    getMaleEmployeesCount(): number {

        return this.employees.filter(e => e.gender === 'Male').length;

    }

    getFemaleEmployeesCount(): number {

        return this.employees.filter(e => e.gender === 'Female').length;

    }

    // Depending on which radio button is selected, this method updates

    // selectedEmployeeCountRadioButton property declared above

    // This method is called when the child component (EmployeeCountComponent)

    // raises the custom event - countRadioButtonSelectionChanged

    // The event binding is specified in employeeList.component.html

    onEmployeeCountRadioButtonChange(selectedRadioButtonValue: string): void {

        this.selectedEmployeeCountRadioButton = selectedRadioButtonValue;

    }

}

<tr \*ngFor="let employee of employees;"

    \*ngIf="selectedEmployeeCountRadioButton=='All'

    || selectedEmployeeCountRadioButton==employee.gender">

The above line of code raises the following error  
Can't have multiple template bindings on one element. Use only one attribute named 'template' or prefixed with \*.  
  
**employeeList.component.html**

<employee-count [all]="getTotalEmployeesCount()"

                [male]="getMaleEmployeesCount()"

                [female]="getFemaleEmployeesCount()"

                (countRadioButtonSelectionChanged)="onEmployeeCountRadioButtonChange($event)">

</employee-count>

<br /><br />

<table>

    <thead>

        <tr>

            <th>Code</th>

            <th>Name</th>

            <th>Gender</th>

            <th>Annual Salary</th>

            <th>Date of Birth</th>

        </tr>

    </thead>

    <tbody>

        <ng-container \*ngFor="let employee of employees;">

            <tr \*ngIf="selectedEmployeeCountRadioButton=='All' ||

                       selectedEmployeeCountRadioButton==employee.gender">

                <td>{{employee.code | uppercase}}</td>

                <td>{{employee.name | employeeTitle:employee.gender }}</td>

                <td>{{employee.gender}}</td>

                <td>{{employee.annualSalary | currency:'USD':true:'1.3-3'}}</td>

                <td>{{employee.dateOfBirth | date:'dd/MM/y'}}</td>

            </tr>

        </ng-container>

        <tr \*ngIf="!employees || employees.length==0">

            <td colspan="5">

                No employees to display

            </td>

        </tr>

    </tbody>

</table>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Interfaces in Angular 2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Angular services tutorial

**Why do we need a service in Angular**  
A service in Angular is generally used when you need to reuse data or logic across multiple components. Anytime you see logic or data-access duplicated across multiple components, think about refactoring that piece of logic or data-access code into a service. Using a service ensures we are not violating one of the Software principles - DRY ((Don't repeat yourself). The logic or data access is implemented once in a service, and the service can be used across all the components in our application.   
  
Without the service you would have to repeat your code in each component. Imagine the overhead in terms of time and effort required to develop, debug, test and maintain the duplicated code across multiple places instead of having that duplicated code at one central place like a service and reusing that service where required.  
  
**Creating a service in Angular :** We will be working with the same example that we have been working with so far in this video series. Add a new TypeScript file to the "employee" folder and name it employee.service.ts. Copy and paste the following code. At the moment we have the data hard-coded in the service method. In a later video we will discuss retrieving data from a remote server using HTTP.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Angular and ASP.NET Web API