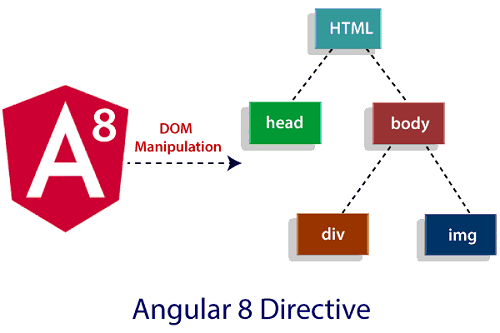


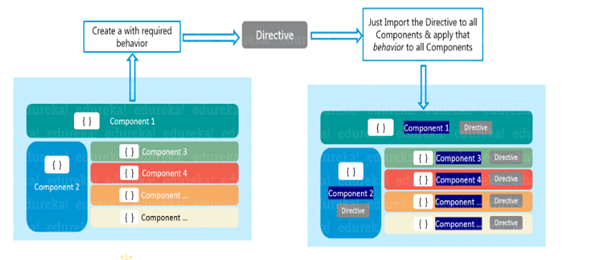
Angular directives are classes that add additional behavior to elements in your Angular applications. Using directives you can change the appearance or behavior of an element, change the DOM layout by adding and removing DOM elements.

The Angular directives are used to manipulate the DOM. By using Angular directives, you can change the appearance, behavior or a layout of a DOM element. It also helps you to extend HTML



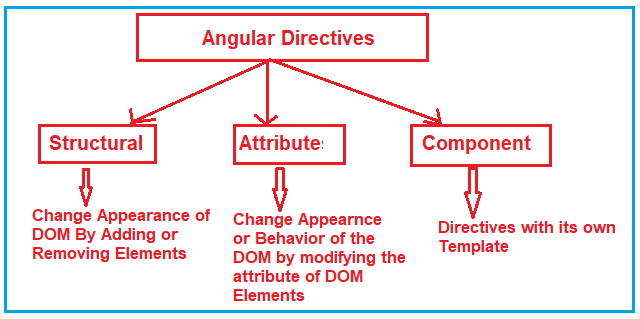
Why we require Directives?

you want a similar functionality in all the components for an example fade-in and fade-out functionality, you can take two approaches. The common approach would be, you can explicitly write the code in all the components for the required behavior, but it would be tedious and complex. Alternatively, like a function in a programming language, you can write the code and later you can call it anytime whenever you want that behavior of that function. Similarly, you can create a directive and write the behavior inside it. Then, wherever you need that behavior, you can import the directive.

ngular Directive is basically a class with a @Directive decorator. You might be wondering what are decorators? ***Decorators*** are functions that modify JavaScript classes. Decorators are used for attaching metadata to classes, it knows the configuration of those classes and how they should work.

You would be surprised to know that a component is also a directive-with-a-template. A @Component decorator is actually a @Directive decorator extended with template-oriented features. Whenever Angular renders a directive, it changes the DOM according to the instructions given by the directive. Directive appears within an element tag similar to attributes.

The Angular Directive can be classified into two types: ***structural***and ***attribute***directives.



**Angular Component Directives**

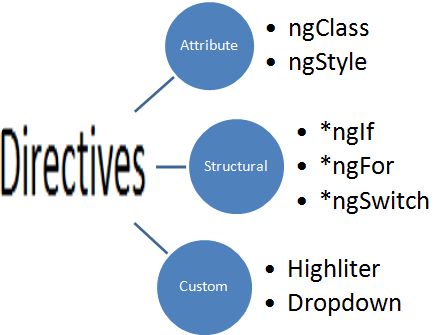
Component directives form the main class. It possesses the details about how the component should be instantiated, processed and utilized at runtime.

**Angular Structural Directives**

As far as a structure directive is concerned, it is associated with manipulating the dom elements. You will find an **asterisk (\*) prefix** before the directive. We can take **\*ngFor** and **\*ngIf** as examples here.

**Angular Attribute Directives**

When it comes to altering the behavior and look of the dom element, you turn to attribute directives. You can come up with your own directive as shown in the example below:



##### **Structural Directives:**

The Structural Directives are responsible for the HTML layout. That means, they will shape or reshape the HTML view by simply adding or removing the elements from the DOM. These directives are basically used to handle how the component or the element should render in a template.

In Angular, there are three structural directives are available. They are as follows:

1. **NgFor (\*ngFor)**
2. **NgIf (\*ngIf)**
3. **NgSwitch (\*ngSwitch)**

##### **Attribute Directives:**

Attribute Directives are basically used to modify the behavior or appearance of the DOM element or the Component. In Angular, there are two in-built attribute directives available. They are as follows:

1. **NgStyle**: This NgStyle Attribute Directive is basically used to modify the element appearance or behavior.
2. **NgClass**: This NgClass Attribute Directive is basically used to change the class attribute of the element in the DOM or in the Component to which it has been attached.

We will discuss these two built-in attributes in detail in our upcoming articles.

##### **Component Directives:**

We have already discussed Component in our previous articles. The Component is also a type of directive in angular with its own template, styles, and logic needed for the view. The Component Directive is the most widely used directive in the angular application and you cannot create an angular application without a component.

A component directive requires a view along with its attached behavior and this type of directive adds DOM Elements. The Component Directive is a class with **@Component** decorator function.

The naming convention for components is **name.component.ts**. For example, if you want to create a component with the name student then it should be student.component.ts.

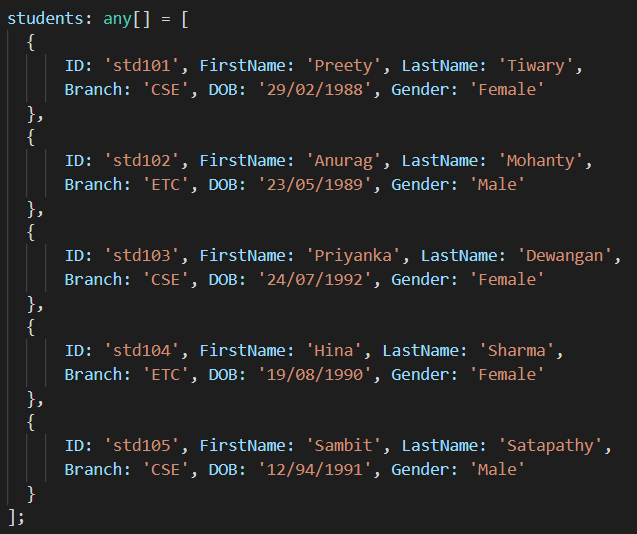
## ****Angular ngFor Directive with Examples****

##### **What is Angular ngFor Directive?**

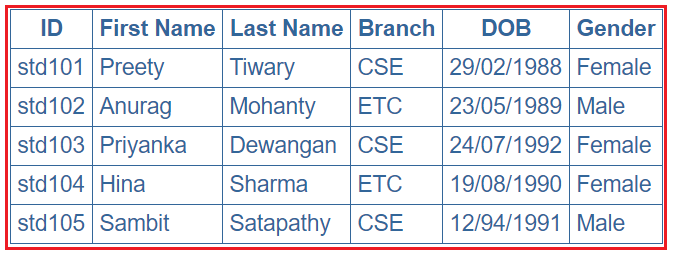
* The built-in ngFor directive belongs to the Structural directive category.
* As it belongs to the structural directive category,
* it is used to change the structure of the DOM.
* The ngFor directive is very much similar to the “for loop” used in most of the programming languages.
* So, the NgFor directive is used to iterate over a collection of data
* . The syntax to use ngFor directive is:
* **\*ngFor=”let <value> of <collection>”**
* Here, <value> is a variable name and collection is a property on your component which a collection of data.
* Usually an array but it can be anything that can be iterated over in a for loop.

##### **Understanding Angular ngFor Directive:**

Let us understand the ngFor structural directive in angular application with an example. We are going to use the following array of Student objects in this demo.



Then we want to display the above students in a table on a web page as shown below.



Let’s discuss the Step By Step Procedure to achieve the above output using Angular ngFor Directive:

##### **Step1: Modify the app.component.ts file**

Open **app.component.ts** file and then copy and paste the following code in it. You can find this file within the app folder of your angular project.

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'DataBinding';

  greet(){

    event?.stopPropagation();

    alert("innerelement");

  }

  greet1(){

    alert("OuterElement");

  }

ngforExample

app.component.ts

  students: any[] = [

    {

    ID: 'std101', FirstName: 'Preety', LastName: 'Tiwary',

    Branch: 'CSE', DOB: '29/02/1988', Gender: 'Female'

    },

    {

    ID: 'std102', FirstName: 'Anurag', LastName: 'Mohanty',

    Branch: 'ETC', DOB: '23/05/1989', Gender: 'Male'

    },

    {

    ID: 'std103', FirstName: 'Priyanka', LastName: 'Dewangan',

    Branch: 'CSE', DOB: '24/07/1992', Gender: 'Female'

    },

    {

    ID: 'std104', FirstName: 'Hina', LastName: 'Sharma',

    Branch: 'ETC', DOB: '19/08/1990', Gender: 'Female'

    },

    {

    ID: 'std105', FirstName: 'Sambit', LastName: 'Satapathy',

    Branch: 'CSE', DOB: '12/94/1991', Gender: 'Male'

    }

    ];

}

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

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

import { AppRoutingModule } from './app-routing.module';

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

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

**app.module.ts**

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

FormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

As you can see in the above code, we have created one array (i.e. students) which a collection of student data. Again we have specified the templateUrl and StyleUrls, so, let proceed and modifies these two files.

##### **Step2: Modify app.component.css file**

Open **app.component.css** file and then copy and paste the following code in it. You can find this file within your app folder. The following styles are going to be used in our HTML page to style table data.

table {

    color: #369;

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

    font-size: large;

    border-collapse: collapse;

}

td {

    border: 1px solid #369;

    padding: 5px;

}

th {

    border: 1px solid #369;

    padding: 5px;

}

##### **Step3: Modify app.component.html file**

Open **app.component.html** file and then copy and paste the following code in it. You can also find this file within your app folder.

<table>

    <thead>

        <tr>

            <th>ID</th>

            <th>First Name</th>

            <th>Last Name</th>

            <th>Branch</th>

            <th>DOB</th>

            <th>Gender</th>

        </tr>

    </thead>

    <tbody>

        <tr \*ngFor='let student of students'>

            <td>{{student.ID}}</td>

            <td>{{student.FirstName}}</td>

            <td>{{student.LastName}}</td>

            <td>{{student.Branch}}</td>

            <td>{{student.DOB}}</td>

            <td>{{student.Gender}}</td>

        </tr>

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

            <td colspan="7">

                No Students to display

            </td>

        </tr>

    </tbody>

</table>

##### **Understand the above Code:**

1. Here, the **ngFor**directive is used to iterate over a collection. In this example, the collection is an array of students.
2. As the **ngFor**directive is a structural directive, so it is prefixed with **\* (star)**. So, the point that you need to remember is, all the structural directive are prefixed with a \*.
3. **\*ngFor=’let student of students’** – In this statement, the ‘**student**‘ is called template input variable, which can be accessed by the <tr> element and any of its child elements.
4. The **ngIf**structural directive displays the row “**No Students to display**” when the student’s property does not exist or when there are no students in the array. We will discuss this directive in detail in our upcoming articles.

##### **Step4: Modify app.module.ts file**

Open **app.module.ts** file which is present inside the app folder and then copy and paste the following code.

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

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

**import** **{** AppRoutingModule **}** from './app-routing.module';

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

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

@NgModule**({**

declarations: **[**

AppComponent

**]**,

imports: **[**

BrowserModule,

AppRoutingModule,

FormsModule

**]**,

providers: **[]**,

bootstrap: **[**AppComponent**]**

**})**

**export** **class** AppModule **{** **}**

That’s it. We have done with our implementation. Now run the application and you will notice that the students are displayed in the table as expected.

##### **ngFor – Local Variables:**

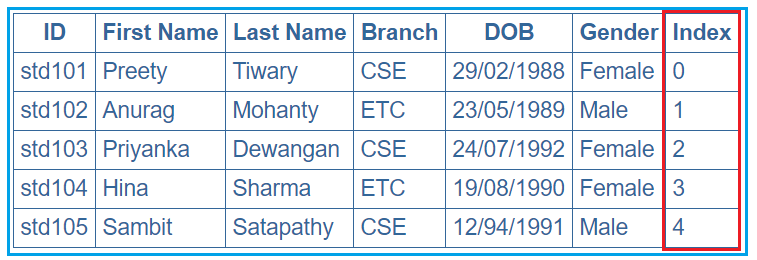
The ngFor structural directive has the following local variable which you can use to customize the data.

1. **Index**: This variable is used to provide the index position of the current element while iteration.
2. **First**: It returns boolean true if the current element is the first element in the iteration else it will return false.
3. **Last**: It returns boolean true if the current element is the last element in the iteration else it will return false.
4. **Even**: It returns boolean true if the current element is even element based on the index position in the iteration else it will return false.
5. **Odd**: It returns boolean true if the current element is an odd element based on the index position in the iteration else it will return false.

Let us understand the above ngFor local variables one by one with example.

##### **How to get the index of an item in a collection in an angular application?**

We can get the index of an item in a collection using the index property of the ngFor directive. Let us understand this with an example. What we want to do here is, along with the student data, we also want to display the index position as shown in the below image.



In order to achieve this, we want to use the ngFor local variable index. So, modify the app.component.html as shown below.

**<table>**

**<thead>**

**<tr>**

**<th>**ID**</th>**

**<th>**First Name**</th>**

**<th>**Last Name**</th>**

**<th>**Branch**</th>**

**<th>**DOB**</th>**

**<th>**Gender**</th>**

**<th>**Index**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr** \*ngFor='let student of students; let i=index'**>**

**<td>**{{student.ID}}**</td>**

**<td>**{{student.FirstName}}**</td>**

**<td>**{{student.LastName}}**</td>**

**<td>**{{student.Branch}}**</td>**

**<td>**{{student.DOB}}**</td>**

**<td>**{{student.Gender}}**</td>**

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

**</tr>**

**</tbody>**

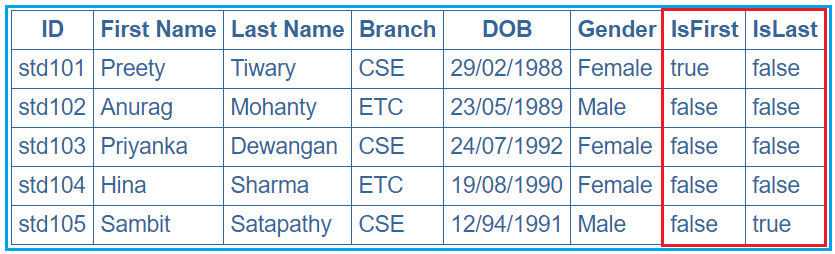
**</table>**

Notice that in the above code, we are using the index property of the Angular ngFor directive to store the index position of an element in a template input variable “i”. The variable “i” is then used in the <td> element where we want to display the index value. We used the let keyword to create the template input variable “i”.  The let keyword in angular is very much similar to the var keyword.

The index of an element is extremely useful when you are creating the HTML elements dynamically. We will discuss creating the HTML elements dynamically in our upcoming articles.

##### **How to identify the first and the last elements in a collection in an angular application?**

To identify the First and Last elements in a collection, you need to use the first and last properties of the ngFor directive respectively. Let us understand this with an example. We want to display the student data along with whether that student is the first or the last student as shown in the below image.



In order to achieve this, modify the **app.component.html** file as shown below.

**<table>**

**<thead>**

**<tr>**

**<th>**ID**</th>**

**<th>**First Name**</th>**

**<th>**Last Name**</th>**

**<th>**Branch**</th>**

**<th>**DOB**</th>**

**<th>**Gender**</th>**

**<th>**IsFirst**</th>**

**<th>**IsLast**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr** \*ngFor='let student of students; let isFirst = first; let isLast = last'**>**

**<td>**{{student.ID}}**</td>**

**<td>**{{student.FirstName}}**</td>**

**<td>**{{student.LastName}}**</td>**

**<td>**{{student.Branch}}**</td>**

**<td>**{{student.DOB}}**</td>**

**<td>**{{student.Gender}}**</td>**

**<td>**{{isFirst}}**</td>**

**<td>**{{isLast}}**</td>**

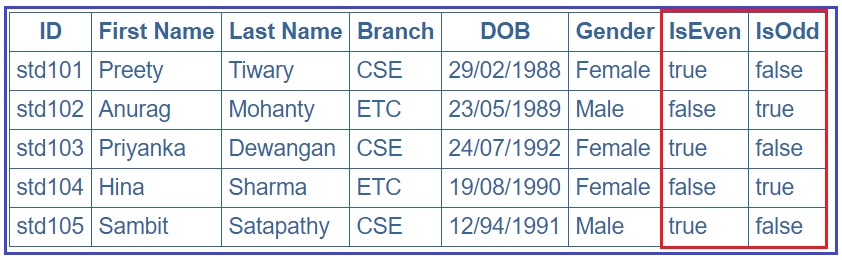
**</tr>**

**</tbody>**

**</table>**

##### **How to identify the even and odd elements in a collection in an angular application?**

In order to identify the Event and Odd elements in a collection in angular, you need to use the even and odd local variable of the ngFor directive respectively. Let us understand this with an example. We want to display the student data along with whether that student is odd or even as shown in the below image.



In order to achieve this, modify the **app.component.html** file as shown below.

**<table>**

**<thead>**

**<tr>**

**<th>**ID**</th>**

**<th>**First Name**</th>**

**<th>**Last Name**</th>**

**<th>**Branch**</th>**

**<th>**DOB**</th>**

**<th>**Gender**</th>**

**<th>**IsEven**</th>**

**<th>**IsOdd**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr** \*ngFor='let student of students; let isEven = even; let isOdd = odd'**>**

**<td>**{{student.ID}}**</td>**

**<td>**{{student.FirstName}}**</td>**

**<td>**{{student.LastName}}**</td>**

**<td>**{{student.Branch}}**</td>**

**<td>**{{student.DOB}}**</td>**

**<td>**{{student.Gender}}**</td>**

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

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

**</tr>**

**</tbody>**

**</table>**

##### **Why do we need the Angular ngFor trackBy?**

The use of trackBy is to improve the performance of the angular application. It is usually not needed by default but needed only when your application running into performance issues.

The Angular ngFor directive may perform poorly with the large collections.  A small change to the collection such as adding a new item or removing an existing item from the collection may trigger a cascade of DOM manipulations.

Suppose, we have some data coming from some API and we are storing these data into some kind of collection like an array and then we need to update these data over the webpage using ngFor directive. By default, what angular framework will do is, it will remove all the DOM elements that are associated with the data and will create them again in the DOM tree even if the same data is coming. That means a lot of DOM Manipul

##### **Example to understand ngFor trackBy in Angular Application:**

Let us understand the need for angular ngFor trackBy with an example step by step.

##### **Step1: Modify app.component.ts file as shown below.**

Open app.component.ts file and then copy and paste the following in it. As you can see in the below code, the constructor of the AppComponent class initializes the student’s collection property with 3 students’ objects. On the other hand, the **getStudents()** method of the AppComponent class update the same student’s collection property with 5 student objects (the 3 existing students plus another two new student object).

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

@Component**({**

selector: 'app-root',

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

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

**})**

**export** **class** AppComponent **{**

students: **any[]**;

**constructor()** **{**

this.students = **[**

**{**

ID: 'std101', FirstName: 'Santosh', LastName: 'Jena', Branch: 'CSE',

DOB: '29/02/1988', Gender: 'Male'

**}**,

**{**

ID: 'std102', FirstName: 'Anurag', LastName: 'Mohanty', Branch: 'ETC',

DOB: '23/05/1989', Gender: 'Male'

**}**,

**{**

ID: 'std103', FirstName: 'Priyanka', LastName: 'Dewangan', Branch: 'CSE',

DOB: '24/07/1992', Gender: 'Female'

**}**,

**]**;

**}**

getStudents**()**: **void** **{**

this.students = **[**

**{**

ID: 'std101', FirstName: 'Santosh', LastName: 'Jena', Branch: 'CSE',

DOB: '29/02/1988', Gender: 'Male'

**}**,

**{**

ID: 'std102', FirstName: 'Anurag', LastName: 'Mohanty', Branch: 'ETC',

DOB: '23/05/1989', Gender: 'Male'

**}**,

**{**

ID: 'std103', FirstName: 'Priyanka', LastName: 'Dewangan', Branch: 'CSE',

DOB: '24/07/1992', Gender: 'Female'

**}**,

**{**

ID: 'std104', FirstName: 'Hina', LastName: 'Sharma', Branch: 'ETC',

DOB: '19/08/1990', Gender: 'Female'

**}**,

**{**

ID: 'std105', FirstName: 'Sambit', LastName: 'Satapathy', Branch: 'CSE',

DOB: '12/94/1991', Gender: 'Male'

**}**

**]**;

**}**

**}**

##### **Step2: Modify the app.component.css file**

We are going to use some styles in our HTML page. So, open **app.component.css** file and then copy and paste the following code in it.

table **{**

color: *#369*;

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

font-size: large;

border-collapse: collapse;

**}**

td **{**

border: 1px solid *#369*;

padding:5px;

**}**

th**{**

border: 1px solid *#369*;

padding:5px;

**}**

##### **Step3: Modify app.module.ts file**

Open your root module i.e. **app.module.ts** file and then copy and paste the following code in it.

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

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

**import** **{** AppRoutingModule **}** from './app-routing.module';

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

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

@NgModule**({**

declarations: **[**

AppComponent

**]**,

imports: **[**

BrowserModule,

AppRoutingModule,

FormsModule

**]**,

providers: **[]**,

bootstrap: **[**AppComponent**]**

**})**

**export** **class** AppModule **{** **}**

##### **Step4: Modify app.component.html file**

Open **app.component.html** file and then copy and paste the following code in it.

**<table>**

**<thead>**

**<tr>**

**<th>**ID**</th>**

**<th>**First Name**</th>**

**<th>**Last Name**</th>**

**<th>**Branch**</th>**

**<th>**DOB**</th>**

**<th>**Gender**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr** \*ngFor='let student of students'**>**

**<td>**{{student.ID}}**</td>**

**<td>**{{student.FirstName}}**</td>**

**<td>**{{student.LastName}}**</td>**

**<td>**{{student.Branch}}**</td>**

**<td>**{{student.DOB}}**</td>**

**<td>**{{student.Gender}}**</td>**

**</tr>**

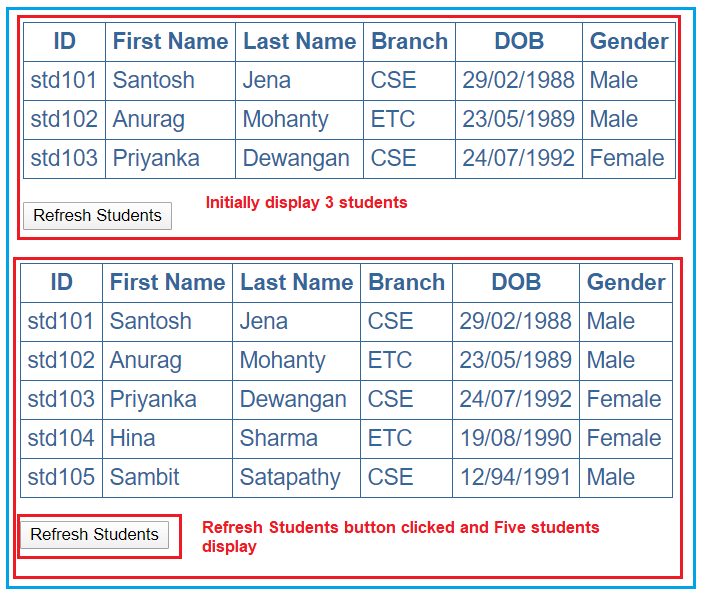
**</tbody>**

**</table>**

**<br** **/>**

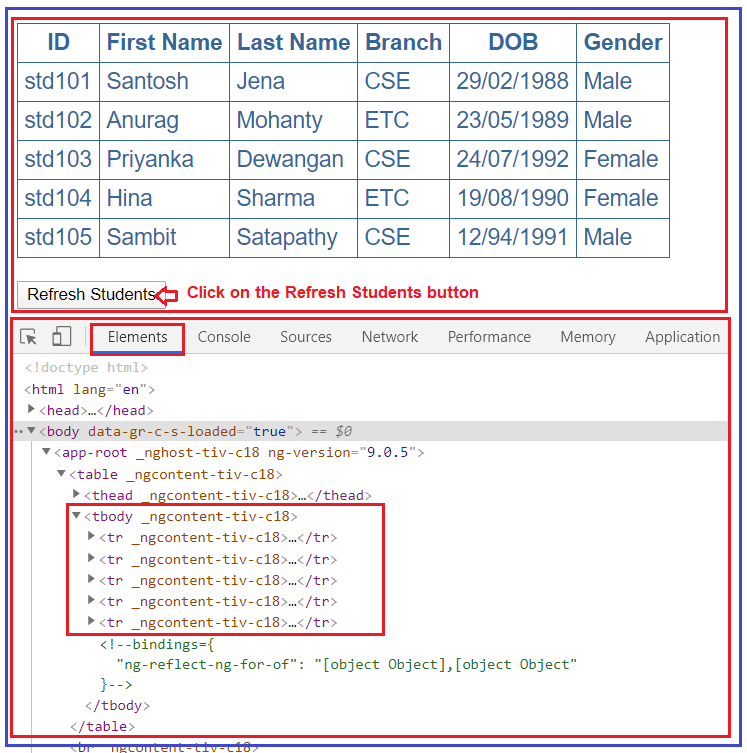
**<button** (click)='getStudents()'**>**Refresh Students**</button>**

In the above HTML, at the moment we are not using the **trackBy** with **ngFor** directive. So when the page loads for the first time you will see 3 students which are initialized through the constructor of the AppComponent class and when you click on the “**Refresh Students**” button, then you will see the 4th and 5th students as well as shown in the below image.



Now you may think that it just added the additional rows for the 4th and 5th students but that’s not true. It internally destroyed all the <tr> and <td> elements of all the students and then recreated them. To confirm this launch the browser developer tools by pressing the F12 key. Then click on the “Elements” tab and expand the <table> and then <tbody> elements.

At this point click on the “**Refresh Students**” button and you will notice that all the <tr> elements are briefly highlighted indicating that they are destroyed and recreated as shown in the below image.



This is because Angular Framework by default keeps track of the objects using the object references. So, when you click on the **“Refresh Students”** button, it will get different object references and as a result, Angular has no other choices but to delete all the old DOM elements and insert the new DOM elements. Image what could happen if the data size is huge.

##### **How to solve the above problem?**

In order to solve the above performance issue problem, the angular framework provides one function called trackBy which will help us to track the items which have been added or removed. The trackBy function will take two arguments, first is the index and the second one is the current item and it will return one unique identifier as return a value using which we can track that item. In our example, we are going to track by Student ID as the student id is unique for each student.

In order to use trackBy, add the following method in the app.component.ts file.

**trackByStudentID(index: number, student: any): string {**  
**return student.ID;**  
**}**

Then do the following changes in the **app.component.html** file:

**<tr \*ngFor=’let student of students; trackBy:trackByStudentID’>**

Here we are using the trackBy along with ngFor directive. At this point, run the application and then launch the browser developer tools by pressing the F12 key. When you click on the **“Refresh Students”** button for the first time, then you can notice that only the 4th and 5th rows of the students are highlighted indicating that only those <tr> elements are added.

On the subsequent clicks, nothing is highlighted meaning none of the <tr> elements are destroyed or added as the students’ collection has not changed. Each and every time when you click on the **“Refresh Students”** button you get different object references, but as the Angular is now tracking the student objects using the Student Id instead of object references, as a result, the respective DOM elements are not affected.

##### **ngular ngIf Directive:**

The ngIf is a structural directive and it is used to add or removes the HTML element and its descendant elements from the DOM layout at runtime conditionally. That means it conditionally shows the inline template.

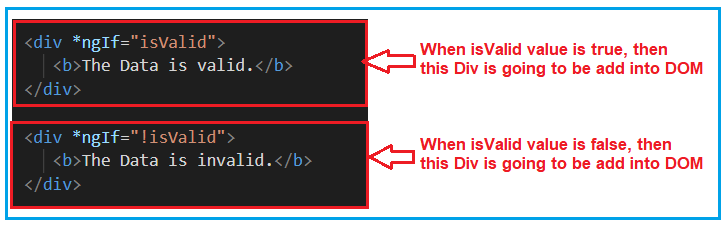
The ngIf directive works on the basis of a boolean true and false results of a given expression. If the condition is true, the elements will be added into the DOM layout otherwise they simply removed from the DOM layout.

The basic syntax of the ngIf directive is very simple, all we need to do is prefix the directive name with an asterisk (\*) as shown below and add it anywhere inside your template. Here if the given “expression” result is false or null, then the elements HTML element and its descendant elements will not be added to the DOM.

**\*ngIf = “expression”**

##### **Example to understand Angular ngIf Directive:**

Please have a look at the following image. As you can, here we have two div Element and we are using the ngIf directive to add or remove the div from the DOM. If the isValid (this is a variable defined in the component class) value is true, then the first div is going to be added into the DOM and if the value is false, then the second div is going to be added into the DOM.



So, first, modify the **app.component.html** file as shown below.

**<div** \*ngIf="isValid"**>**

**<b>**The Data is valid.**</b>**

**</div>**

**<div** \*ngIf="!isValid"**>**

**<b>**The Data is invalid.**</b>**

**</div>**

Then modify the **app.component.ts** file as shown below.

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

@Component**({**

selector: 'app-root',

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

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

**})**

**export** **class** AppComponent **{**

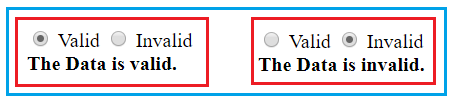
isValid: **boolean** = **true**;

**}**

At this moment, change the value of isValid Boolean property (true and false) and check the output in the browser and based on the value, the respective div will be added into the DOM and that you can in the webpage.

##### **Another Example For Better Understanding:**

Let us enhance the same example. Now, instead of hardcoding the isValid Boolean property value, we will set this value from the radio buttons. We will create two radio buttons and based on the selected radio button value we will set the IsValid property value. For better understanding please have a look at the below image.



When the Valid radio button is checked, we need to add the first Div to the DOM and when the InValid radio button is checked we need to add the second Div into the DOM. Let’s see how to implement this.

##### **Modify the app.component.ts file:**

Please modify the app.component.ts file as shown below. By default, we set the Boolean isvalid property to true. We have also created one method i.e. ChangeData which accepts a Boolean parameter (in this case valid) and then it set the isValid property value with the incoming parameter value (valid).

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

@Component**({**

selector: 'app-root',

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

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

**})**

**export** **class** AppComponent **{**

isValid: **boolean** = **true**;

ChangeData**(**valid: **boolean)** **{**

this.isValid = valid;

**}**

**}**

##### **Modify the app.component.html file:**

Please modify the **app.component.html** file as shown below. Here we have created two radio buttons with the name rb. Then we attach our method ChangeData to the click event of the radio button. For the Valid radio button, we are passing true to the ChangeData method and for the Invalid radio button, we are passing false to the ChangeData method.

**<div>**

**<input** type="radio" name="rb" (click)= "ChangeData(true)" checked **>** Valid

**<input** type="radio" name="rb" (click)= "ChangeData(false)"**>** Invalid

**</div>**

**<div** \*ngIf="isValid"**>**

**<b>**The Data is valid.**</b>**

**</div>**

**<div** \*ngIf="!isValid"**>**

**<b>**The Data is invalid.**</b>**

**</div>**

With the above changes in place, now run the application and you should get the output as expected.

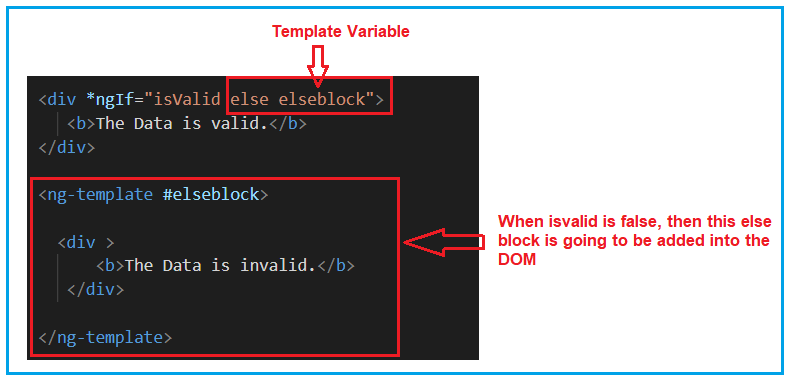
##### **Angular NgIf directive with else block:**

Like other programming languages such as C, Java, and C#, the else is also used when you want to display something for the false condition of NgIf block. The syntax to use the ngIf block with the else block as follows:

**<div \*ngIf = “condition; else elseBlock”>…</div>**  
**<ng-template #elseblock>….</ng-tempalte>**

Here, for else block you need to use <ng-template> element. It is referred by a template reference variable. NgIf will use that template reference variable to display the else block when the condition is false.

Let us work with the same example. Please have a look at the following image. In the else block we have created a template variable with the name elseblock (you can give any name as per your choice). So, what will happen here is, it will check the ngIf condition and if the condition is true then it will execute if block. On the other hand, if the condition is false, then it will look for the else block. As we created the else block with the name elseblock, so it searches for ng-template tag followed by template variable name (<ng-template #elseblock>) and add that block to the DOM.



So, modify the **app.component.html** file as shown below and then see the output in the browser.

**<div>**

**<input** type="radio" name="rb" (click)= "ChangeData(true)" checked **>** Valid

**<input** type="radio" name="rb" (click)= "ChangeData(false)"**>** Invalid

**</div>**

**<div** \*ngIf="isValid else elseblock"**>**

**<b>**The Data is valid.**</b>**

**</div>**

**<ng-template** #elseblock**>**

**<div** **>**

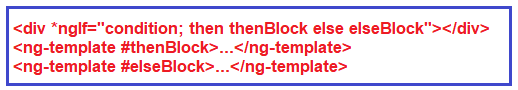
**<b>**The Data is invalid.**</b>**

**</div>**

**</ng-template>**

##### **NgIf with Then and else:**

The NgIf with then and else is used as follows. When the condition is true then the <ng-template> with the reference variable then block is executed and when the condition is false then the <ng-template> with the reference variable else block is executed. For better understanding please have a look at the following image.



We can have more than one <ng-template> for then and else block and at runtime, we can switch to that ng-template by changing the value of then block and else block. At any given point of time, one ng-template is going to be executed.

So, modify the **app.component.html** file as shown below and then see the output in the browser.

**<div>**

**<input** type="radio" name="rb" (click)= "ChangeData(true)" checked **>** Valid

**<input** type="radio" name="rb" (click)= "ChangeData(false)"**>** Invalid

**</div>**

**<div** \*ngIf="isValid then thenblock else elseblock"**>** **</div>**

**<ng-template** #thenblock**>**

**<div>**

**<b>**The is Then Block**</b>**

**</div>**

**</ng-template>**

**<ng-template** #elseblock**>**

**<div** **>**

**<b>**The is Else Block**</b>**

**</div>**

**</ng-template>**

##### **What is the Angular ngSwitch directive?**

The Angular ngSwitch directive is actually a combination of two directives i.e. an attribute directive and a structural directive. It is very similar to the switch statement of other programming languages like Java and C# but within a template.

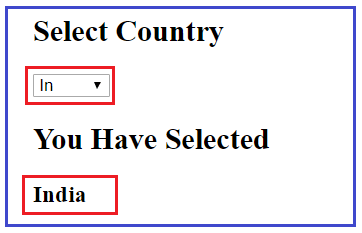
The ngSwitch directive lets you hide or show the HTML elements based on an expression. Here, you can also define a default section using the ng-switch-default directive to show a section if no other sections get a match. So, while working with ngSwitch directive, you need three things to keep in mind, they are ngSwitch, ngSwitchCase and ngSwithDefault.

##### **Example to understand Angular ngSwitch Directive:**

Let us understand the Angular ngSwitch Directive with an example. What we are going to do is, we will provide a dropdown list to the user to select the country code and based on the selected country code we will provide the full country name. Whenever no country code is selected then it will display you have not selected any country. For example, when you have not selected the country code, then by default the select option is selected from the drop dropdown list and You have not selected any country is displayed as sown in the below image.



But when selected a particular country code let say IN, then India should display as shown in the below image.



Let us see how to implement the above using Angular ngSwitch directive step by step.

##### **Step1: Modify the app.component.ts file**

First, modify the **app.component.ts** file as shown below. As you can see here, we have created one property i.e. **dropDownValue** and one method i.e. SetDropDownValue. And we are setting the property value using the methods parameter value. This method is going to be called by the dropdown list **change** event.

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

@Component**({**

selector: 'app-root',

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

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

**})**

**export** **class** AppComponent **{**

**public** dropDownValue = "";

SetDropDownValue**(**drpValue : **any)** **{**

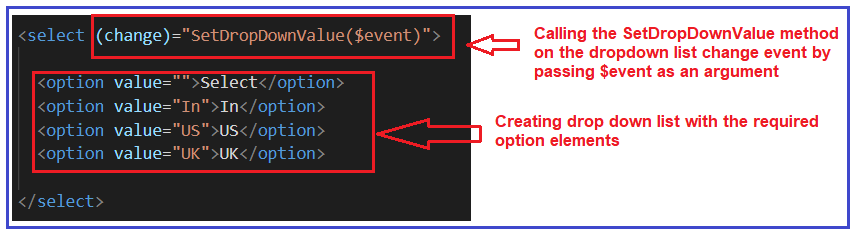
this.dropDownValue = drpValue.target.value;

**}**

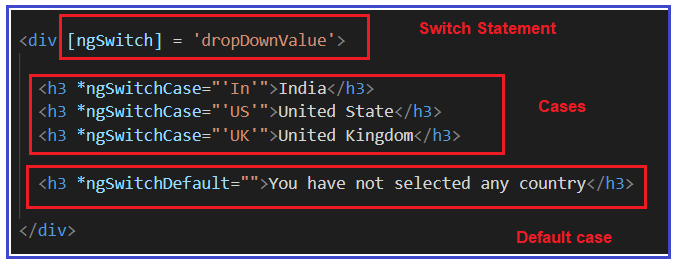
**}**

##### **Step2: Modify the app.component.html file**

This is the important part and here you need to understand two things. First, we need to create the drop-down list and then on the change event of the dropdown list, we need to call the **SetDropDownValue** method that we created within our component class. For better understanding please have a look at the following image.



In the second step, we need to use the ngSwitch directive and in the ngSwitch directive, we need to bind the property (**dropDownValue**) and whose value is set by the drop-down list change event. Then we need to write the required ngSwitchCase statements. As we have three options in the drop-down list and so we have one ngSwitchCase one per each option value. Finally, we have the default switch case i.e. ngSwitchDefault which will execute when no options are selected from the dropdown list. For better understanding please have a look at the following image.



So, modify the **app.component.html** file as shown below and check in the browser and it should work as expected.

**<h2>**Select Country**</h2>**

**<select** (change)="SetDropDownValue($event)"**>**

**<option** value=""**>**Select**</option>**

**<option** value="In"**>**In**</option>**

**<option** value="US"**>**US**</option>**

**<option** value="UK"**>**UK**</option>**

**</select>**

**<h2>**You Have Selected**</h2>**

**<div** [ngSwitch] = 'dropDownValue'**>**

**<h3** \*ngSwitchCase="'In'"**>**India**</h3>**

**<h3** \*ngSwitchCase="'US'"**>**United State**</h3>**

**<h3** \*ngSwitchCase="'UK'"**>**United Kingdom**</h3>**

**<h3** \*ngSwitchDefault=""**>**You have not selected any country**</h3>**

**</div>**