DATA BINDING

Data Binding is available right from AngularJS, Angular 2 and is now available in Angular 4 as well. We use curly braces for data binding - {{}}; this process is called interpolation. We have already seen in our previous examples how we declared the value to the variable title and the same is printed in the browser.

The variable in the **app.component.html** file is referred as {{title}} and the value of title is initialized in the **app.component.ts** file and in **app.component.html**, the value is displayed.

Let us now create a dropdown of months in the browser. To do that , we have created an array of months in **app.component.ts** as follows −

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

title = 'Angular 4 Project!';

// declared array of months.

months = ["January", "Feburary", "March", "April", "May",

"June", "July", "August", "September",

"October", "November", "December"];

}

The month’s array that is shown above is to be displayed in a dropdown in the browser. For this, we will use the following line of code −

<!--The content below is only a placeholder and can be replaced. -->

<div style="text-align:center">

<h1>

Welcome to {{title}}.

</h1>

</div>

<div> Months :

<select>

<option \*ngFor="let i of months">{{i}}</option>

</select>

</div>

EVENT BINDING

In this chapter, we will discuss how Event Binding works in Angular 4. When a user interacts with an application in the form of a keyboard movement, a mouse click, or a mouseover, it generates an event. These events need to be handled to perform some kind of action. This is where event binding comes into picture.

Let us consider an example to understand this better.

app.component.html

<!--The content below is only a placeholder and can be replaced.-->

<div style = "text-align:center">

<h1>

Welcome to {{title}}.

</h1>

</div>

<div> Months :

<select>

<option \*ngFor = "let i of months">{{i}}</option>

</select>

</div>

<br/>

<div>

<span \*ngIf = "isavailable; then condition1 else condition2">

Condition is valid.

</span>

<ng-template #condition1>Condition is valid</ng-template>

<ng-template #condition2>Condition is invalid</ng-template>

</div>

<button (click)="myClickFunction($event)">

Click Me

</button>

In the **app.component.html** file, we have defined a button and added a function to it using the click event.

Following is the syntax to define a button and add a function to it.

(click)="myClickFunction($event)"

The function is defined in the **.ts** file: **app.component.ts**

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

title = 'Angular 4 Project!';

//array of months.

months = ["January", "Feburary", "March", "April",

"May", "June", "July", "August", "September",

"October", "November", "December"];

isavailable = true;

myClickFunction(event) {

//just added console.log which will display the event details in browser on click of the button.

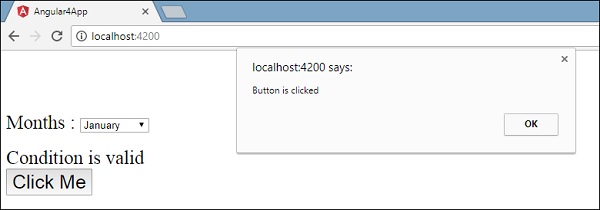
alert("Button is clicked");

console.log(event);

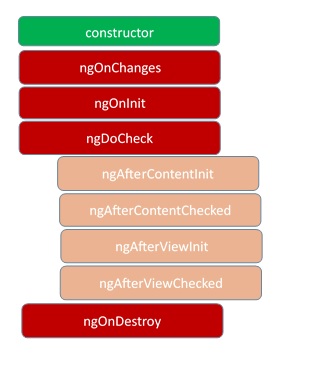
}

}

Upon clicking the button, the control will come to the function **myClickFunction** and a dialog box will appear, which displays **the Button is clicked** as shown in the following screenshot −



**Life Cycle Method Of Angular**

In Angular, every component has a life-cycle, a number of different stages it goes through from initializing to destroying. There are 8 different stages in the component lifecycle. Every stage is called  life cycle hook events. So, we can use these hook events in different phases of our application to obtains fine controls on the components. Since a component is a typescript class, for that reason every component must have a constructor method. The constructor of the component class executes first before the execution of any other life cycle hook event. If we need to inject any dependencies into the component, then the constructor is the best place to inject that dependency. After executing the constructor, Angular executes its life cycle hook methods in a specific order.  
  


These stages are mainly divided into two phases – one is linked to the component itself and another is linked to the children of that component.

* ***ngOnChanges***

This event executes every time when a value of an input control within the component has been changed. Actually, this event fires first when a value of a bound property has been changed. It always receives a changed data map, containing the current and previous value of the bound property wrapped in a SimpleChange.

* 1. {"name":{"previousValue":"","currentValue":"Amit"}}

In this example above, one value change has been detected to the input property name. The value of this property has been changed from an empty string to a string value “Amit”.

* ***ngOnInit***This event initializes after angular first displays the data-bound properties or when the component has been initialized. This event basically is called only once just after the ngOnChanges() events. This event is mainly used for initializing data in a component.
* ***ngDoCheck***This event is triggered every time when the input properties of a component are checked. We can use this hook method to implement the check with our own logic check. Basically, this method allows us to implement our own custom change detection logic or algorithm for any component.
* ***ngAfterContentInit***This lifecycle method is executed when Angular performs any content projection within the component views. This method executes only for the first time when all the bindings of the component need to be checked for the first time. This event executes just after the ngDoCheck() method. This method is basically linked with the child component initializations.
* ***ngAfterContentChecked***This lifecycle hook method executes every time when the content of the component has been checked by the change detection mechanism of the Angular. This method called after ngAfterContentInit() method. This method is also called on every subsequent execution of ngDoCheck(). This method is also mainly linked with the child component initializations.
* ***ngAfterViewInit***This lifecycle hook method executes when the component’s view has been fully initialized. This method is initialized after Angular initializes the component’s view and child views. It is called only the first time after ngAfterContentChecked(). This lifecycle hook method only applies to components.
* ***ngAfterViewChecked***This method is just called after the ngAterViewInit() method. It is executed every time the when the view of the given component has been checked by the change detection algorithm of Angular. This method executes every subsequence execution of the ngAfterContentChecked(). This method also executes when any binding of the children directives has been changed. So this method is very useful when the component waits for some value which is coming from its child components.
* ***ngOnDestroy***This method will be executed just before Angular destroys the components. This method is very useful for unsubscribing the observables and detaching the event handlers to avoid memory leaks. Actually, it is called just before the instance of the component is finally destroyed. This method called only once just before the component is removed from the DOM.