ANGULAR 10

Presented by

Eng./Abanoub Nabil

Teaching assistant at ITI

Agenda

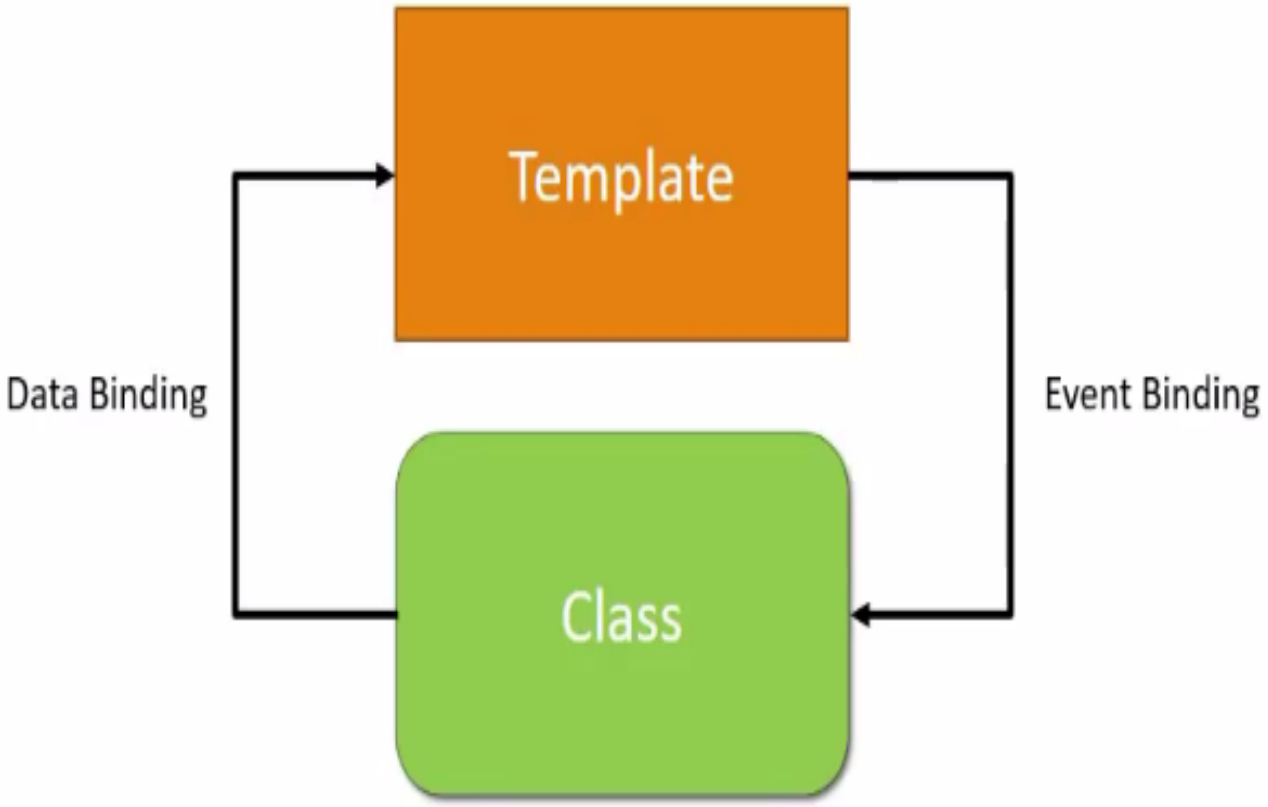
- Event Binding
- Template reference variable
- Two way Binding
- Structural directives
- Pipes

Event Binding

- Previously we discussed **data binding** that flow from component class to component template but sometimes to respond to user events such as mouse click we need the data flow in the other direction **from component template to component class.**

  - **This is called event binding.**

Event Binding(cont.)



Event Binding(cont.)

- **Let’s see example:**

```
<input type="button" value="Greating" (click)=“Greating ()">

public greatingMsg="";

Greating()
{
      console.log("Hello Abanoub");
      this.greatingMsg="welcome abanoub";

}
```

Event Binding(cont.)

- Sometimes you want to know information about the event itself.
- You will use **$event** that will give information about the event itself.

```
<input type="button" value="Greating"
  (click)="myfunc2($event)">
```

Template reference variable

- When there is a user interaction you might some data to flow from the view to the class to perform an operation.

- To easily access **DOM elements** and there properties Angular provide this by **Template reference variable.**

Template reference variable(cont.)

- Let say that we have an input element

```
<input  #myInput type="text"/>
```

and we have button

```
<button >log</button>
```

What we want is when the user click on the button show what value inside the text field.

```
<button (click)="logMsg(myInput.value)"></button>
```

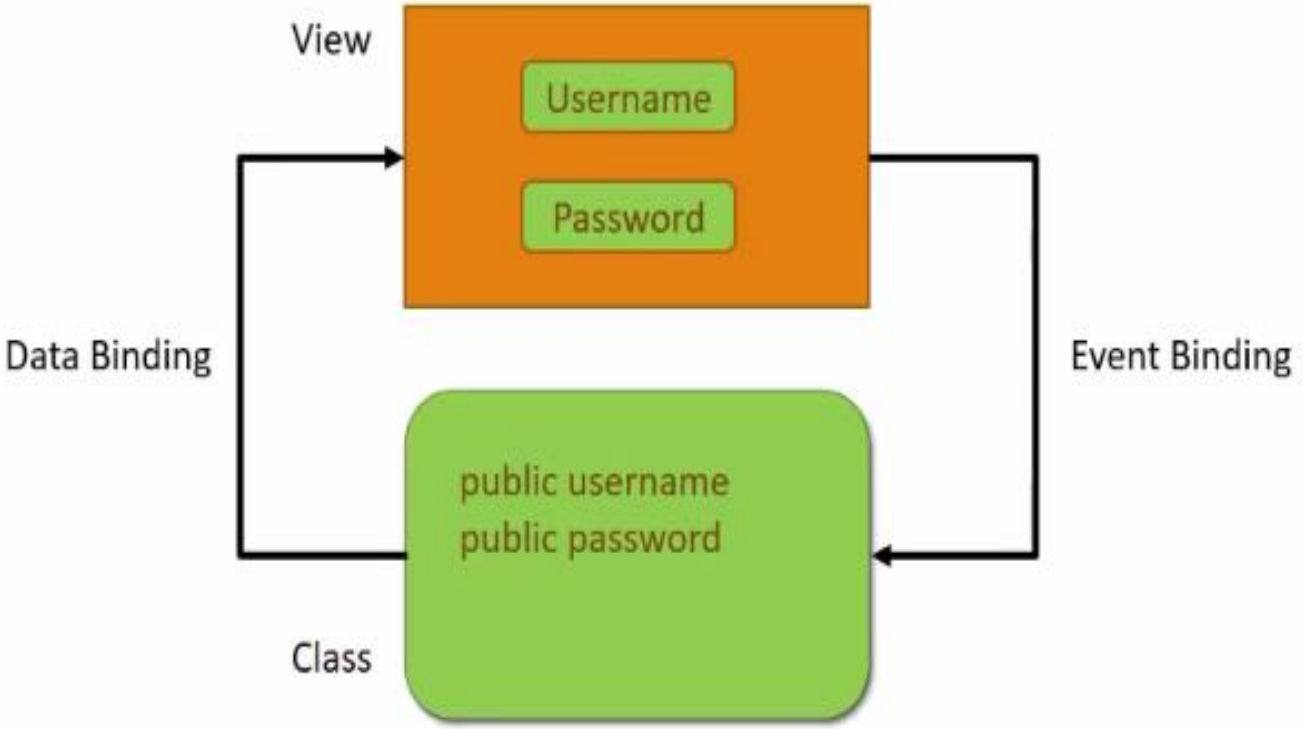
Two way Binding

- When you work with form elements its essential that your **model and view become in think**.

- **For example** consider a login form that has username and password and the component class has the properties username and class.

- When you update form the properties in the component class should automatically updated and versa.

Two way Binding (cont.)



Two way Binding(cont.)

- ***Allow us to update property and in the same time display the value of the property.***
- For two way binding Angular provide another directive which is **ngModel** directive.
- Angular is not aware with **ngModel** directive because it is in a separate module called **formsModule** so we must import it.
- Open **app.module.ts** and import it

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

- and then add it to the imports array.

Two way Binding(cont.)

- Two way binding example

```
<input [(ngModel)]="name" type="text" >
{{name}}
```

Structural directives

- **Add or remove** HTML elements from the DOM
  - ngIf
  - ngSwitch
  - ngFor

Structural directives (cont.)

- **ngIf**

```
<h2 *ngIf="true" >
    Hello ngIf
</h2>

<h2 *ngIf="displayName" >
    Hello ngIf 2
</h2>
```

Structural directives (cont.)

```
<!--Use ngIf with else-->

<h2 *ngIf="displayIfOrElse else elseBlock" >
      This is the if body
</h2>

<ng-template #elseBlock>
<h2>
     This is else body
</h2>
</ng-template>
```

Structural directives (cont.)

```
<!--Another syntax to *ngIf-->

<div *ngIf="displayName; then thenBlock; else ElseBlock">

</div>

<ng-template #thenBlock>
    <h2>
        then block.
    </h2>
</ng-template>

<ng-template #ElseBlock>
    <h2>
        else block.
    </h2>
</ng-template>
```

Structural directives (cont.)

- **ngSwitch**

```
<!--[ngSwitch] directive-->

<div [ngSwitch]= "color">

    <div *ngSwitchCase="'red'">You Picked Red Color</div>
    <div *ngSwitchCase="'blue'">You Picked blue Color</div>
    <div *ngSwitchCase="'green'">You Picked green Color</div>
    <div *ngSwitchDefault>Pick Color again</div>

</div>
```

Note:-color is TypeScript property.

Structural directives (cont.)

- **ngFor**

```
public cars=["Fiat","Lancer","KIA","Optera"];
```

```
<div *ngFor="let car of cars">
  <h2>{{car}}</h2>
</div>
```

Structural directives (cont.)

```
<!--*ngFor directive with index-->

<div *ngFor="let car of cars ; index as i">
    <h2>{{i+" "}} {{car}}</h2>
</div>
```

Structural directives (cont.)

```
<!--*ngFor directive with odd and even-->

<div *ngFor="let car of cars ; odd as o">
   <h2>{{o+" "}} {{car}}</h2>
</div>
```

Structural directives (cont.)

```
<!--*ngFor directive with first and last->

<div *ngFor="let car of cars ; first as f">
   <h2>{{f+" "}} {{car}}</h2>
</div>
```

Pipes

- Allow us to transform data from displaying it in the view.
- See the next Demos examples.

Pipes (cont.)

- Pipes in string.

```
{{name}}
<br/>
{{name |lowercase}}
<br/>
{{name|uppercase}}
<br/>
{{name|titlecase}}
```

Pipes (cont.)

```
{{name|slice:3}}
<!--began after the third character-->
<br/>
{{name|slice:3:5}}
<!--began in the third character to 5 so it will take
4,5-->
```

Pipes(cont.)

- Number Pipes

```
<!--Number Pipes-->
  {{2.678 | number:'1.2-3'}}
<!--1 is the minimum number of integer digits 2 the minimum number of decimal digits 3 the maximum number of decimal digits -->
<br/>
{{2.678 | number:'3.4-5'}}
<br/>
{{2.678 | number:'3.1-2'}}
<br/>
<!--Percent Pipe-->
{{0.25|percent|currency}}
```

Pipes(cont.)

- Currency Pipe

```
{{0.25|currency}} <!--by defulat it is $-->
<br/>
{{0.25 | currency :'GBP'}}
<br/>
{{0.25 | currency :'EUR'}}
<br/>
{{0.25 | currency :'GBP' :'code'}}
```

Pipes(cont.)

- Date Pipes

```
{{date}}
<br/>
{{date|date:'short' }}
<br/>
{{date|date:'shortDate'}}
<br/>
{{date|date:'shortTime'}}
<br/>
{{date|date:'medium'}}
<br/>
{{date|date:'mediumDate'}}
<br/>
{{date|date:'mediumTime'}}
```

Pipes(cont.)

- Create custom pipe

```
ng g pipe pipe_name
```

- And then implement your logic inside the transform method.

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

@Pipe({
  name: 'square'
})
export class SquarePipe implements PipeTransform {

  transform(value: number): number {
    return value*value;
  }

}
```

Questions

Any Questions?

THANK YOU ☺