Angular Developer 3

Exam ?

Angular Interview Questions & Answers

Tasks

Task 1

NICE GUY AT THE GATE

You enter or leave the classroom and there is a guy that is asking about your name.
He either greets you politely or wishes you to have a nice day.

NICE GUY AT THE GATE

1. Create interface for collecting 'name' and 'family name' - inputs
2. Create interface for greeting and byebye'ing - buttons
3. Display alerts upon either greeting or byebye'ing

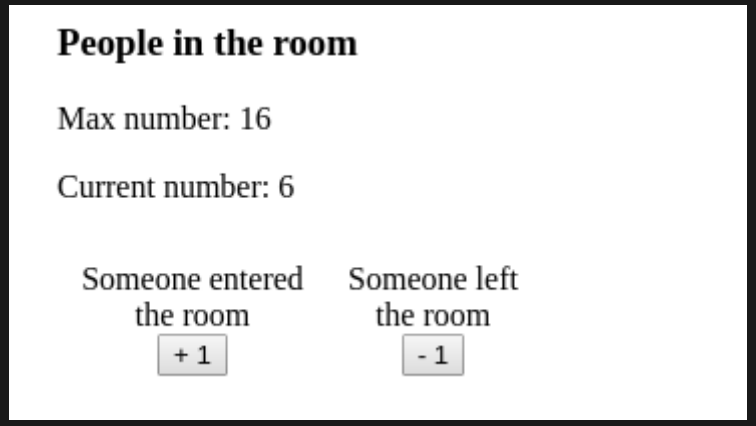
Additionally

- Empty inputs on 'bye bye'

Task 2

PEOPLE IN THE ROOM

Number of people that can be in the room is limited. We need some tool to help us with keeping track of who is coming and who is going...



PEOPLE IN THE ROOM

1. Display max number of people that can be in the room
2. Display current number of people
3. Create interface for increasing and decreasing the current number
    - two buttons (+1 and -1)
4. Display a warning if 'current' is equal or bigger than allowed

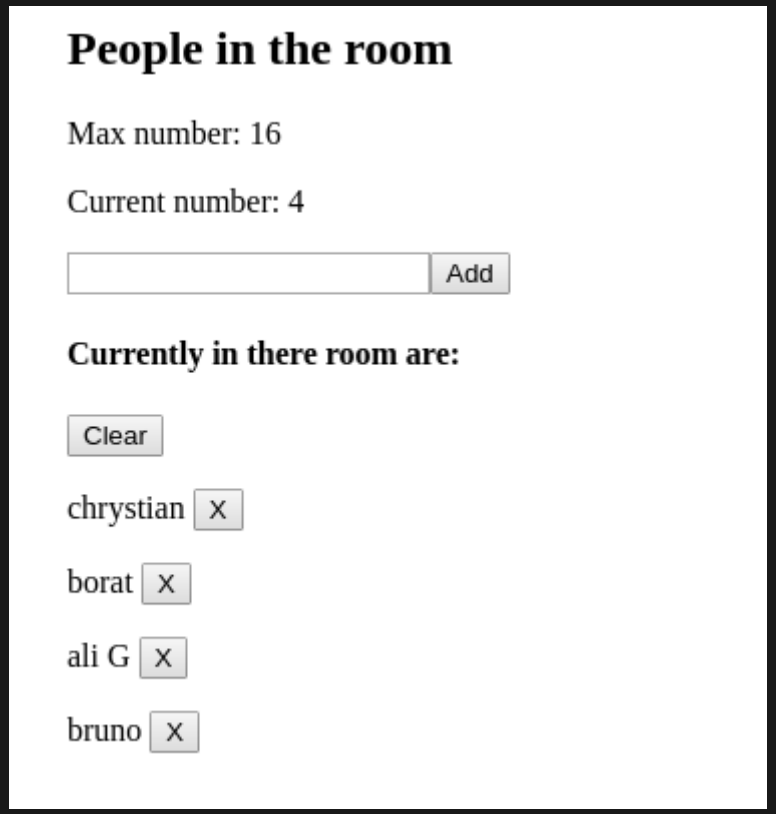
Additionally

- IF user hits +1 BUT there is no more space in the room display an alert
- User colors: when there are 3 seats left orange, when no more seats then red
- Don't allow negative numbers

Task 3

PEOPLE IN THE ROOM V2

We want to keep track of how many people are in the room
BUT ADDITIONALLY know who is there.



PEOPLE IN THE ROOM V2

1. In order to add a person to the list we need to change our interface
    - input field for name
    - button for adding provided name to the list
2. After adding a person to the list clear the input
3. Adjust logic that is checking 'current' vs 'max allowed'
4. Display the list of names
5. It should be possible to remove person from the list
6. It should be possible to completely clear the list

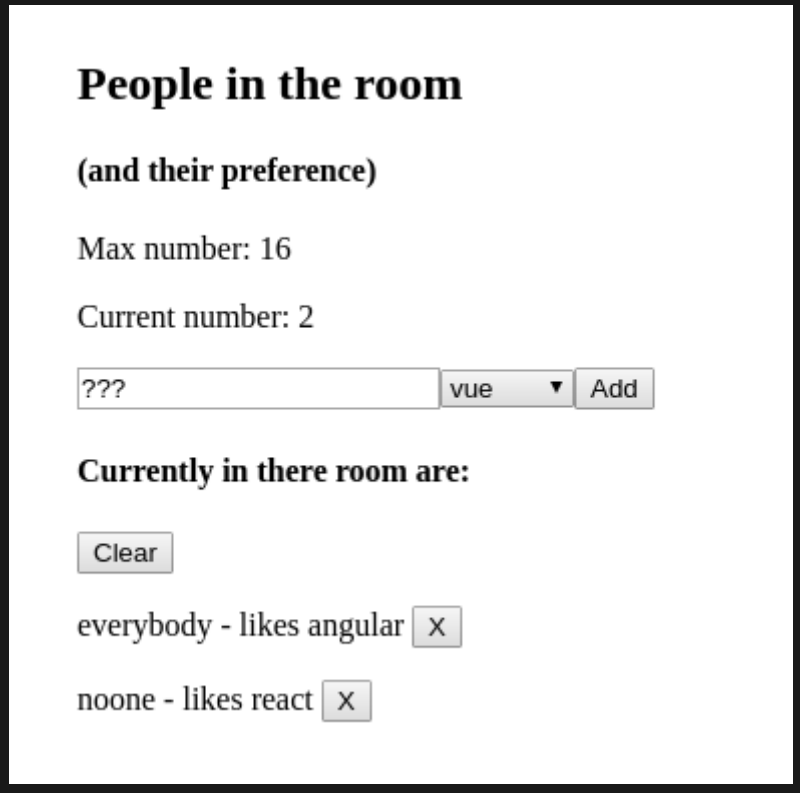
Additionally

- Person on the list needs a name! Prevent adding empty rows.
- Make the list numbered (each person has a number)

Task 4

PEOPLE IN THE ROOM V3 - WHO LIKES WHAT

We need a new feature. In order to know a bit more about people in the room we want some additional info about each of them.



PEOPLE IN THE ROOM V3

1. Upon adding a new person to the list collect info about preferred framework
    - Dropdown with options: angular, react, vue
2. Adjust logic of adding person to the list as now we have
    - name
    - favorite framework name
3. Adjust removing from list logic
4. Adjust list structure and logic to display persons preference

Additionally

- How about adding edit button next to each person on the list?

Template Oriented

Interpolation

```
1 <div>
2   <p>{{ 'Welcome my dear friend' }}</p>
3   <p>{{ 'Ready for a math lesson?' }}</p>
4   <p>{{ '2 + 2 equals ' + (2+2) }}</p>
5 </div>
```

Its basically rendering right?

What else can we interpolate / render ?

```
<div style="{{ 'border: 2px solid black' }}">
    <img src="{{ 'https://place-hold.it/300x200' }}">
</div>
```

```
<div>
    <input value="{{ 'Me is Chrystian' }}"
           placeholder="{{ 'What is your name?' }}"/>
</div>
```

Wait wait... its all static

Template context is...

- template is a part of component
- component has logic and data in a class

...its component instance

Angular building blocks

Modules	
Components	Services
Directives	Interceptors
Pipes	


Modules
Components
Directives
Pipes

```
@Component({
    selector: 'app-my-component',
    template: '<div>{{ title }}</div>',
    styles: [`
        :host {
           margin: 200px;
        }
    `]
})
export class MyComponent {
    title = 'myProject';
}
```

```ts
// src/app/my-component.ts
@Component({
  selector: 'app-my-component',
  templateUrl: './my-component.html',
  styleUrls: ['./my-component.scss']
})
export class MyComponent {
    title = 'myProject';
}
```

```html
<!-- src/app/my-component.html -->
<div>{{ title }}</div>
```

How to use?

```html
<div>
  <app-my-component></app-my-component>
</div>
```

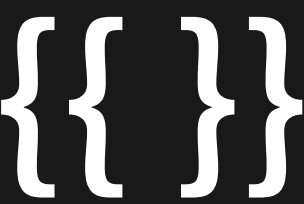
Component (docs)

- Selector - html tag name
- Template - html body
- Logic - ts class
- Styles

(all tied together with @Component decorator)

...back to our templates

Interpolation



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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  public name = 'Chrystian';
}
```

```ts
// src/app/app.component.ts
export class AppComponent {
  public name = 'Chrystian';
}
```

```ts
// src/app/app.component.ts
export class AppComponent {
  public name = 'Chrystian';
}
```

```html
<!-- src/app/app.component.html -->
<div>
  <p>{{ name + ', welcome my dear friend' }}</p>
</div>
```

```ts
// src/app/app.component.ts
export class AppComponent {
  public name = 'Chrystian';
  public avatarUrl = 'https://place-hold.it/300x200';
  public style = 'border: 2px solid black';
}
```

```html
<!-- src/app/app.component.html -->
<div>
  <p>{{ 'Welcome my dear friend' }}</p>
  <div style="{{ styles }}">
      <img src="{{ avatarUrl }}">
  </div>
</div>
```

Template has access to variables...

...component/class methods too

```ts
// src/app/app.component.ts
export class AppComponent {
  public name = 'Chrystian';

  public printWelcome() {
    return this.name + ', welcome...';
  }
}
```

```html
<!-- src/app/app.component.html -->
<div>
  <p>{{ printWelcome() }}</p>
</div>
```

BUT BE CAREFUL WITH THAT

Interpolation

- Printing strings on the screen
- Printing into html attributes (attr vs prop)
- Outputting variables (as strings)
- Doing some simple logic
- Call functions

Good practices

- as simple as possible
- as little function calls as possible
- only print data - don't touch state

Interpolation

Attribute / Property binding

String values

```ts
// src/app/app.component.ts
export class AppComponent {
  public avatarUrl = 'https://place-hold.it/300x200';
}
```

```html
<!-- src/app/app.component.html -->
<div>
  <img [src]="avatarUrl">
</div>
```

Non string values

```ts
// src/app/app.component.ts
export class AppComponent {
  public isDisabled = true;
}
```

```html
<!-- src/app/app.component.html -->
<button [disabled]="isDisabled"><!-- WAY TO GO! -->
  Submit form
</button>
```

Non string values

```ts
// src/app/app.component.ts
export class AppComponent {
  public isReadonly = true;
}
```

```html
<!-- src/app/app.component.html -->
<input [readonly]="isReadonly"><!-- WAY TO GO! -->
```

Css class

```ts
// src/app/app.component.ts
export class AppComponent {
  public classes = 'round-border white-bg';
}
```

```html
<!-- src/app/app.component.html -->
<div [class]="classes">
</div>
```

Css class

```
// src/app/app.component.ts
export class AppComponent {
  public withBorder = true;
  public hasBg = false;

}
```

```
<!-- src/app/app.component.html -->
<div
  [class.round-border]="withBorder"
  [class.white-bg]="hasBg">
</div>
```

Css class

```
// src/app/app.component.ts
export class AppComponent {
  public classes = {
    'round-border': true,
    'white-bg': false
}
```

```
<!-- src/app/app.component.html -->
<div
  [class]="classes">
</div>
```

...classes. But what about styles?

Styles

```
// src/app/app.component.ts
export class AppComponent {
  public styles = 'border: 1px solid red; width: 100px';
}
```

```
<!-- src/app/app.component.html -->
<div
  [style]="styles">
</div>
```

Styles

```
// src/app/app.component.ts
export class AppComponent {
  public styles = 'border: 1px solid red; width: 100px';
}
```

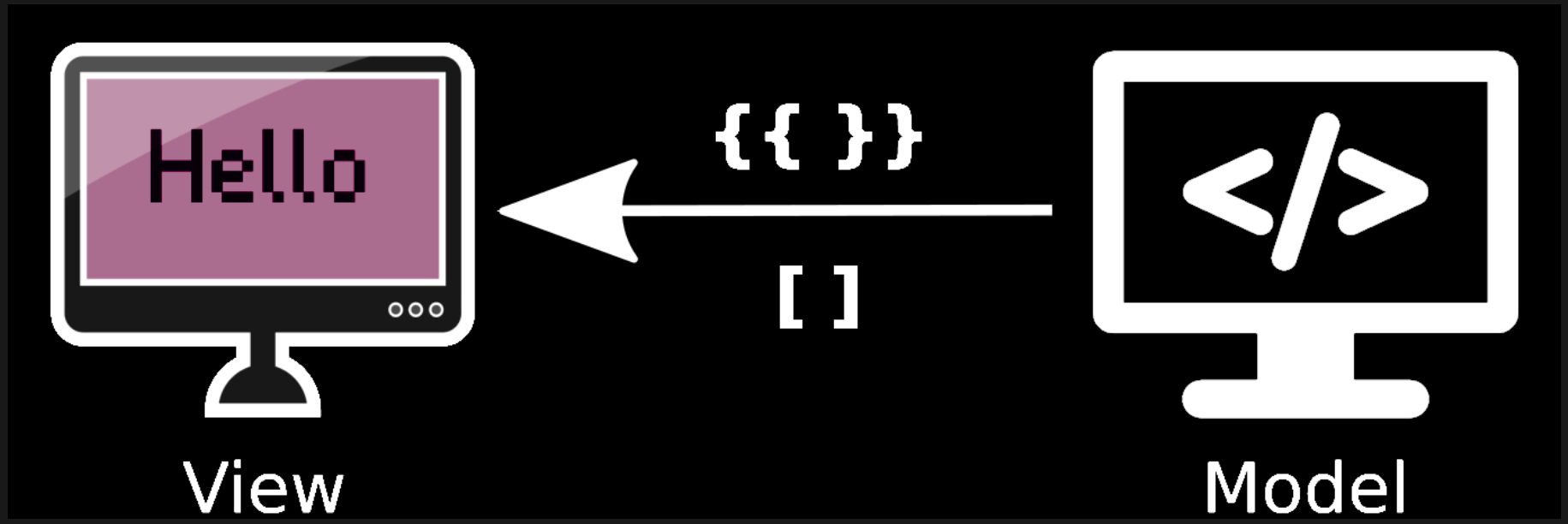
```
<!-- src/app/app.component.html -->
<div
  [style.border]="'1px solid red'"
  [style.width]="'100px'">
</div>
```

Styles

```ts
// src/app/app.component.ts
export class AppComponent {
  public width = window.innerWidth;
}
```

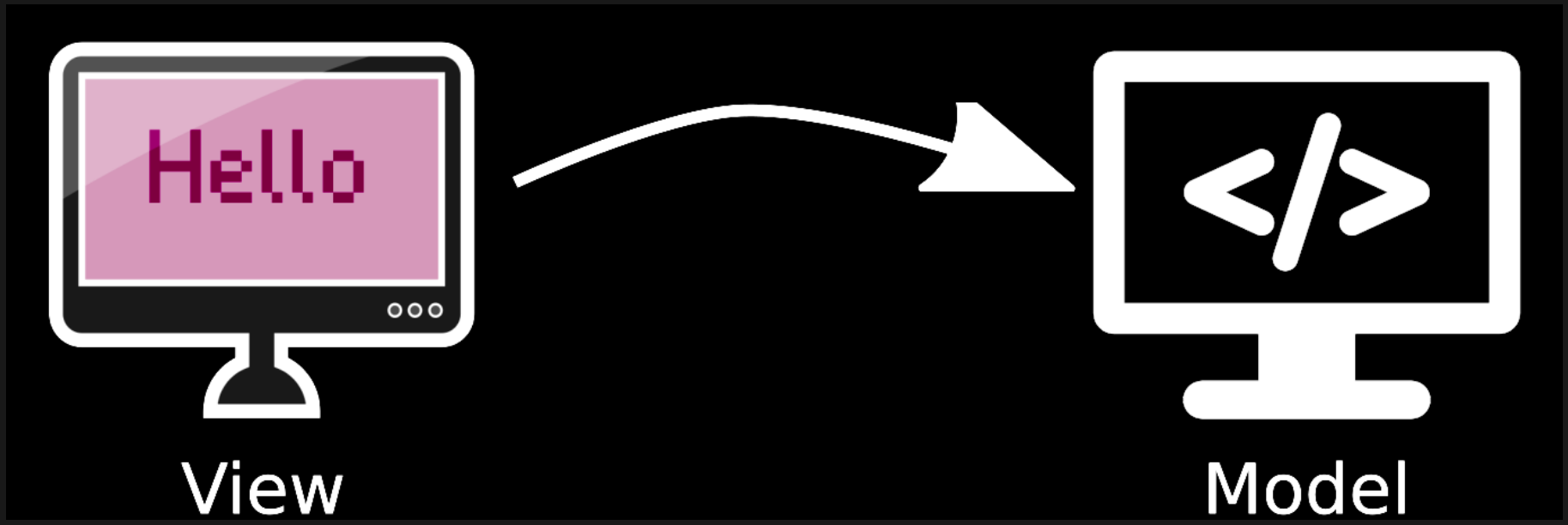
```html
<!-- src/app/app.component.html -->
<div
  [style.border]="'1px solid red'"
  [style.width]="width + 'px'">
</div>
```

Reflecting data in the view



Interpolation or Property Binding

Getting data from user?



Event Binding

( )

Click

```ts
// src/app/app.component.ts
export class AppComponent {
  public submitForm() {
    console.log('submitting...');
  };
}
```

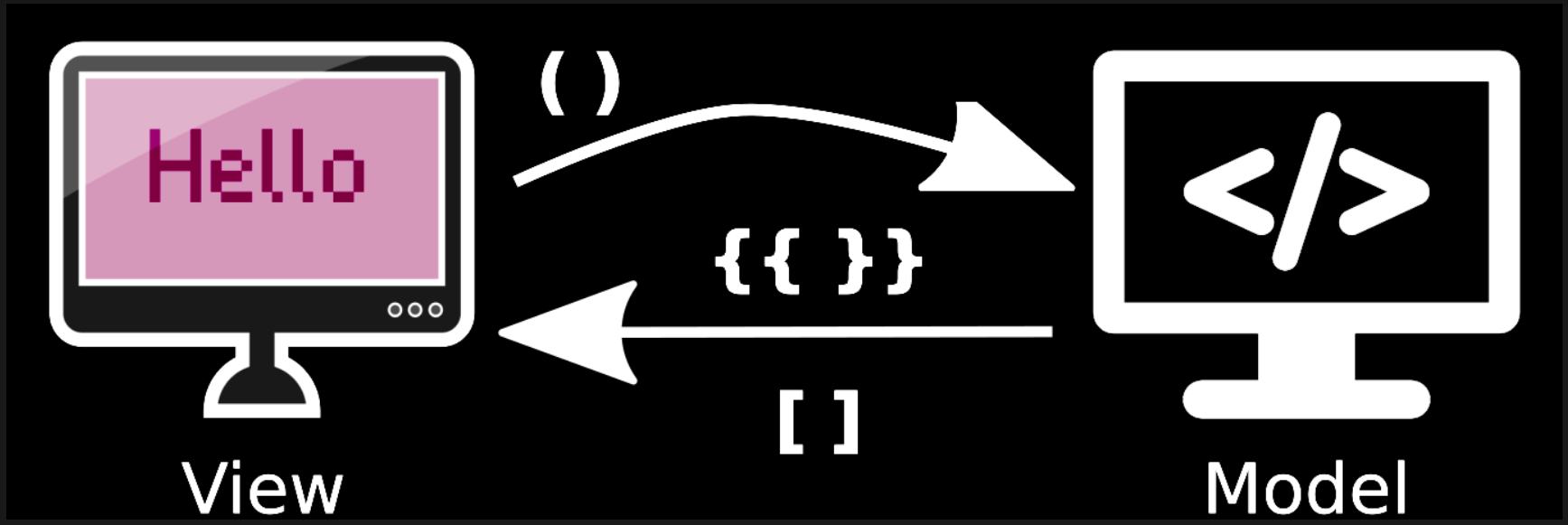
```html
<!-- src/app/app.component.html -->
<button (click)="submitForm()">Save</button>
```

Input

```
// src/app/app.component.ts
export class AppComponent {
  onInputChange(event): void {
    console.log(event.target.value);
  }
}
```

```
<!-- src/app/app.component.html -->
<input (input)="onInputChange($event)">
```

DATA BINDING



TWO WAY DATA BINDING

```
// src/app/app.component.ts
export class AppComponent {
  public name = 'Chrystian';

  onInputChange(event): void {
    this.name = event.target.value;
  }
}
```

```
<!-- src/app/app.component.html -->
Please put your name here:

<input [value]="name"
       (input)="onInputChange($event)"/>
<br/><br/>
You entered: {{ name }}
```

Please put your name here:
Chrystian

You entered: Chrystian

SUMMARY

{{ }}	just prints whatever we put inside
[propName]="variable" [attrName]="variable"	binds variable to specified property or attribute
(eventName)="method($event)"	adds event handler, on event fires method(), can pass the $event object

OK, ONE LAST THING

public vs private

```ts
// src/app/app.component.ts
export class AppComponent {
  public name = 'Chrystian';
  private familyName = 'Ruminowicz';
}
```

```html
<!-- src/app/app.component.html -->
<p>Your name{{ name }}</p>
<p>Your name{{ familyName }}</p>
```

```
error TS2341: Property 'familyName' is private
and only accessible within class 'AppComponent'.
```

More building blocks

Directives

- Component
  (special directive with its own template)
- Attribute - alter behavior or looks
- Structural - change structure

Attribute - ngClass

```
@Component({ ... })
export class MyComponent {
    isVisible = true;
}
```

```
<p [ngClass]="{
    hidden: !isVisible,
    'not-hidden': isVisible
}">
    visible?
</p>
```

Attribute - ngClass

```
@Component({ ... })
export class MyComponent {
    isVisible = true;
    public classes = {
      hidden: !this.isVisible,
      'not-hidden': this.isVisible
    };
}
```

```
<p [ngClass]="classes">
    visible?
</p>
```

Attribute - ngStyle

```
@Component({ ... })
export class MyComponent {
  public styles = {
      color: 'red',
      'font-size': window.innerWidth + 'px'
  };
}
```

```
<p [ngStyle]="styles">
    styled
</p>
```

Attribute - ngModel

!!! IMPORTANT !!!

more in a sec

Other attr directives

- required
- pattern
- routerLink
- formControl
- i18n-title

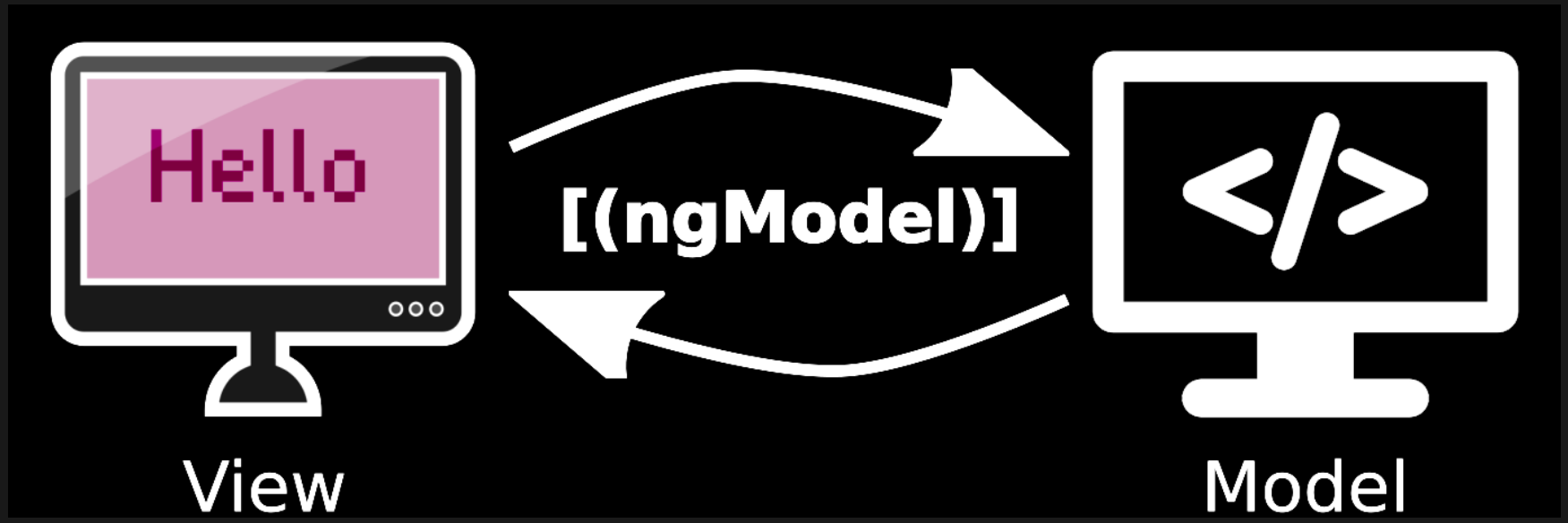
ATTRIBUTE DIRECTIVES (DOCS)

operate on OTHER html elements

and modify their behavior

(we can have many attr directives on a single element)

ngModel


Hello
[(ngModel)]
</>
View
Model

Please put your name here:
Chrystian

You entered: Chrystian

no ngModel :(

```ts
// src/app/app.component.ts
export class AppComponent {
  public name = 'Chrystian';
  onInputChange(event): void {
    this.name = event.target.value;
  }
}
```

```html
<!-- src/app/app.component.html -->
Please put your name here:
<input [value]="name"
       (input)="onInputChange($event)"/>
<br/>
You entered: {{ name }}
```

with ngModel :)

```
@Component({ ... })
export class MyComponent {
  public name = 'Chrystian';
}
```

```
Please put your name here:
<input [(ngModel)]="name"/>
<br/>
You entered: {{ name }}
```

STRUCTURAL DIRECTIVES (DOCS)

modify structure

operate on template - host and child elements

(ONLY one structural directive per html element)

*ngIf

```
@Component({ ... })
export class MyComponent {
    isVisible = true;
}
```

```
<p *ngIf="isVisible">
  visible?
</p>
```

*ngFor

```
@Component({ ... })
export class MyComponent {

    selected = 'bis';
}
```

```
Plans for summer? Select proper word ending.
<select [(ngModel)]="selected">
    <option value="ry">ry</option>
    <option value="bis">bis</option>
</select>
<br/>
---
Cana{{ selected }}
---
```

*ngFor

```
@Component({ ... })
export class MyComponent {
    answers = ['ry', 'bis'];
    selected = 'bis';
}
```

```
Plans for summer? Select proper word ending.
<select [(ngModel)]="selected">
    <option *ngFor="let opt of options"
            [value]="opt">
        {{ opt }}
    </option>
</select>
<br/>
---
Cana{{ selected }}
---
```

*ngIf

*ngIf

```
@Component({ ... })
export class MyComponent {
    isVisible = true;
}
```

```
<p *ngIf="isVisible">
  visible?
</p>
<p *ngIf="!isVisible">
  NOT visible?
</p>
```

*ngIf

```
@Component({ ... })
export class MyComponent {
    color = 'blue'; // red, orange
}
```

```
<p *ngIf="color === 'blue'">
  Blue is the best
</p>
<p *ngIf="color === 'orange'">
  Orange is bester
</p>
<p *ngIf="color === 'red'">
  red
</p>
```

*ngSwitch

```
@Component({ ... })
export class MyComponent {
    color = 'blue'; // red, orange
}
```

```
<div [ngSwitch]="color">
  <p *ngSwitchCase="'blue'">Blue is the best</p>
  <p *ngSwitchCase="'orange'">Orange is bester</p>
  <p *ngSwitchCase="'red'">red the bestest</p>
</div>
```