Data Binding

1. Data binding in detail
2. Two-way binding
3. Decisions and iteration

- Additional techniques

```
Demo app:    AngularDev/Demos/03-DataBinding/DemoApp
To install: npm install
To run:      ng serve
```

Pearson

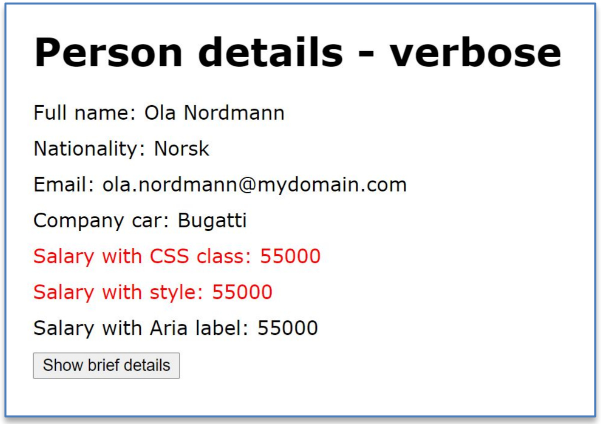
Section 1: Data Binding in Detail

- Example application
- TypeScript code
- Data binding syntax
- Binding to computed values
- Binding target properties
- Binding events
- Safe navigation

Pearson

Example Application

- In this section we dig deeper into data binding syntax
  - In the demo app, click the **Data binding syntax** link
  - Click the button to toggle between verbose / brief modes
  - Notice the button label changes when you click it



Person details - verbose

Full name: Ola Nordmann

Nationality: Norsk

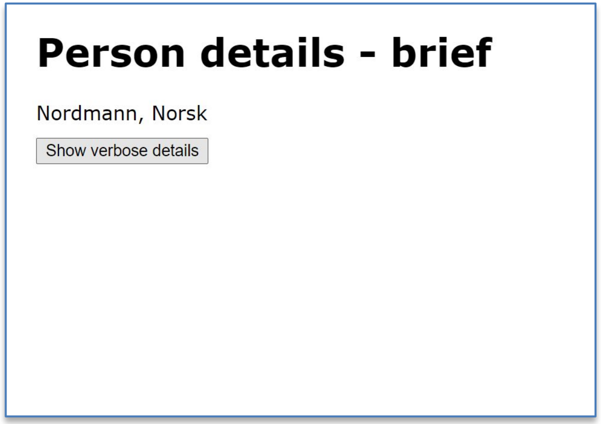
Email: ola.nordmann@mydomain.com

Company car: Bugatti

Salary with CSS class: 55000

Salary with style: 55000

Salary with Aria label: 55000



Person details - brief

Nordmann, Norsk

TypeScript Code

- Here's the partial TypeScript code (discussions follow…)

```
@Component({…})
export class DataBindingSyntaxComponent {
  firstName: string;
  lastName: string;
  nationality: string;
  emailAddress: string;
  companyCar?: Car;
  salary: number;
  verbose: boolean;
  …
  label() {
    return (this.verbose) ? 'Show brief details' : 'Show verbose details';
  }

  onVerboseToggle(event: any) {
    this.verbose = !this.verbose;
  }
}
```

data-binding-syntax.component.ts

Pearson

Data Binding Syntax

- Here's a recap of simple data binding using {{ }}

```
<h1>Person details - verbose</h1>
<div>Full name: {{firstName}} {{lastName}} </div>
<div>Nationality: {{nationality}} </div>
<div>Email: {{emailAddress}} </div>
…                                            data-binding-syntax.component.html
```

- The expressions bind to these component properties:

```
export class DataBindingSyntaxComponent {
  firstName: string;
  lastName: string;
  nationality: string;
  emailAddress: string;
  …
}                                              data-binding-syntax.component.ts
```

Pearson

Binding to Computed Values

- You can bind to computed values within the {{ }}
  - E.g. call a function that returns a value dynamically

```
<button (click)="onVerboseToggle($event)"> {{label()}} </button>
```

```
export class DataBindingSyntaxComponent {
  …
  label() {
    return (this.verbose) ? 'Show brief details' : 'Show verbose details';
  }
}
```

- Template expressions shouldn't change any application state, except the value of the target property

Pearson

Binding Target Properties

- You can bind target properties on an element
  - Enclose property name in `[]`

```
<div [hidden]='!verbose'>
  <h1>Person details - verbose</h1>
  <div>Full name: {{firstName}} {{lastName}} </div>
  <div>Nationality: {{nationality}} </div>
  …
</div>

<div [hidden]='verbose'>
  <h1>Person details - brief</h1>
  <div>{{lastName}}, {{nationality}} </div>
</div>
```

```
export class DataBindingSyntaxComponent {
  …
  verbose: boolean;
}
```

Pearson

Binding Events

- You can bind an event to a function in the component
  - Enclose event name in `()`
  - Call handler function, and pass argument(s) if you like

```
<button (click)="onVerboseToggle($event)"> {{label()}} </button>
```

```
export class DataBindingSyntaxComponent {
  …
  onVerboseToggle(event: any) {
    this.verbose = !this.verbose;
  }
}
```

Safe Navigation

- The safe navigation operator `?.` is very useful
  - Means the field is optional
  - If field is `undefined`, rest of expression is ignored

```
<div>Company car: {{companyCar?.make}}</div>
```

Section 2: Two-Way Data Binding

- 1-way data binding - recap
- 2-way data binding
- Delaying data binding until "done"
- 2-way data binding example
- Supporting 2-way data binding

Pearson

1-Way Data Binding - Recap

- `[]` syntax gives 1-way binding
  - Binds component model to UI property

```
<div [hidden]="verbose">
    …
</div>
```

- `()` syntax gives 1-way binding in the other direction
  - Binds UI event to component event handler

```
<button (click)="onVerboseToggle()">
    …
</button>
```

Pearson

2-Way Data Binding

- `[()]` syntax gives 2-way binding
  - Binds component model to UI, to display data
  - Binds UI to component, to update data in model

```
<input type='number' [(ngModel)]="salary">
```

- Alternatively you can use two separate 1-way bindings
  - `[xxx]` binds component model to UI
  - `(xxxChange)` binds UI to component

```
<input type='number' [ngModel]="salary" (ngModelChange)="salary=$event">
```

Pearson

Delaying Data Binding Until "Done"

- Sometimes you want to delay updating the model value until the user has tabbed out of a text box
  - E.g. a numeric field, don't update model after each digit

- To achieve this, handle the `blur` or `change` events
  - Upon the event, update the model with the UI value

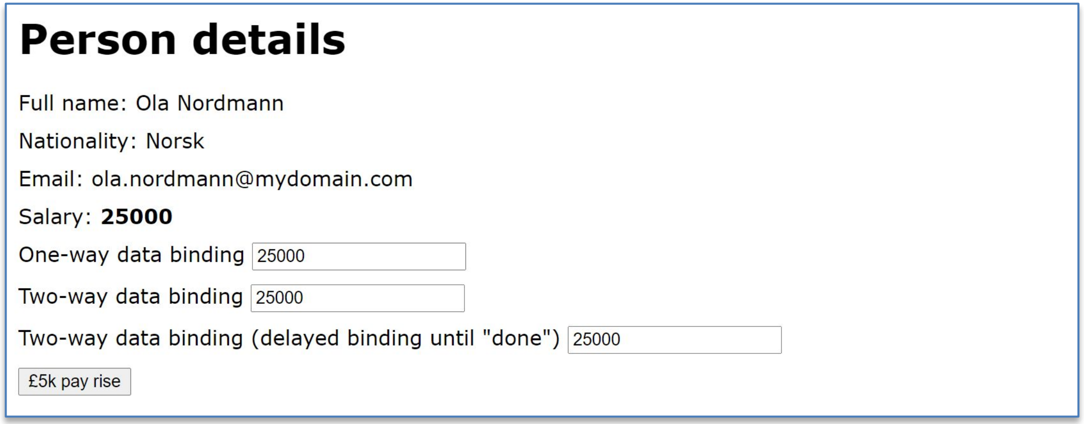
```
<input type='number' [value]="salary" (change)="onSalaryChange($event)">
```

```
export class DataBindingSyntaxComponent {
  …
  onSalaryChange($event) {
    this.salary = Number($event.target.value);
  }
}
```

Pearson

2-Way Data Binding Example (1 of 2)

- Let's see an example of 2-way data binding
  - In the demo app, click the 2-way data binding link
  - There are two text boxes, where you can change salary
  - There's also a button, which gives a £5k pay rise



Person details

Full name: Ola Nordmann

Nationality: Norsk

Email: ola.nordmann@mydomain.com

Salary: **25000**

One-way data binding 25000

Two-way data binding 25000

Two-way data binding (delayed binding until "done") 25000

2-Way Data Binding Example (2 of 2)

- Here are the relevant parts of the HTML and code:

```
Salary: <b> {{salary}} </b>
One-way data binding <input type='number' [value] = "salary">
Two-way data binding <input type='number' [(ngModel)] = "salary">

Two-way data binding (delayed binding until "done")
<input type='number' [value]="salary" (change)="onSalaryChange($event)">

<button (click)="payRise()">Pay rise</button>
```

two-way-binding.component.html

```
export class TwoWayBindingComponent {
  salary: number;
  …
  payRise() {
    this.salary = this.salary + 5000;
  }
  onSalaryChange($event) {
    this.salary = Number($event.target.value);
  }
```

two-way-binding.component.ts

Supporting 2-Way Data Binding (1 of 2)

- Your app dependencies must specify Angular Forms
  - Angular CLI does this for you - see `package.json`

```
"dependencies": {
  "@angular/forms": "~14.1.0",
  …
}
```

package.json

Supporting 2-Way Data Binding (2 of 2)

- Your app module must include `FormsModule`
  - You must modify `app.module.ts` as follows:

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

@NgModule({
  declarations: […],
  imports: [FormsModule, BrowserModule, AppRoutingModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

app.module.ts

Section 3: Decisions and Iteration

- Overview
- If tests
- If-then-else tests
- For loops
- Switches

Overview

- Angular has structural directives that allow you to make decisions and to do iteration in your HTML template
  - `ngIf` – conditional rendering
  - `ngFor` – iterative rendering
  - `ngSwitch` – branch-based rendering

- Aside:
  - In Angular terminology, a *structural directive* is a class that changes the DOM by adding/removing elements

If Tests (1 of 2)

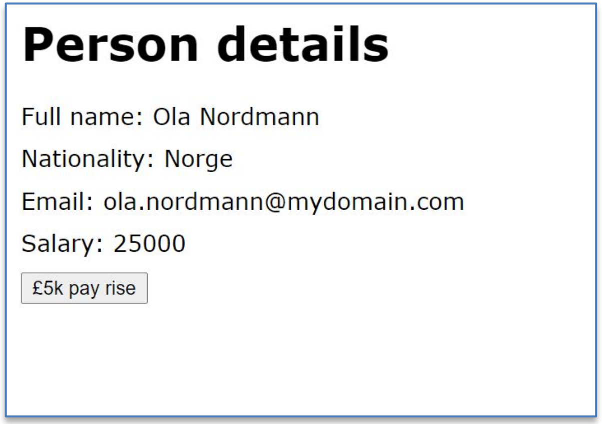
- You can use `ngIf` to perform a test
  - Use the `ngIf` directive, prefixed with `*`
  - Angular embeds content in an `<ng-template>` tag
  - Angular activates `<ng-template>` tag conditionally

```
<div *ngIf="salary > 40000" style="color:red">
  [Upper tax payable on {{salary - 40000}}]
</div>                                        ng-if.component.html
```

If Tests (2 of 2)

- Let's see an example of `ngIf` tests
  - In the demo app, click the **ngIf** link

Initial display



Person details

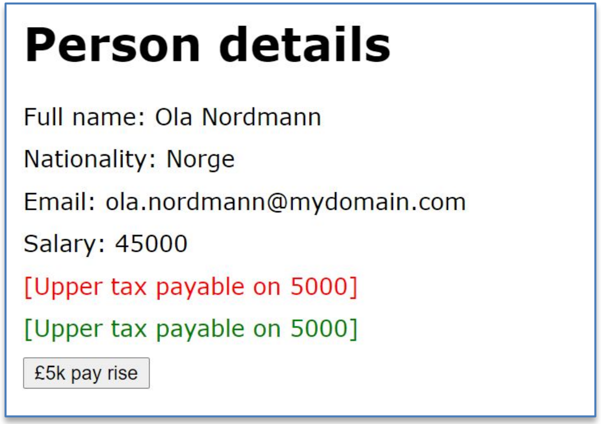
Full name: Ola Nordmann

Nationality: Norge

Email: ola.nordmann@mydomain.com

Salary: 25000

After several pay rises



Person details

Full name: Ola Nordmann

Nationality: Norge

Email: ola.nordmann@mydomain.com

Salary: 45000

[Upper tax payable on 5000]

[Upper tax payable on 5000]

If-Then-Else Tests (1 of 3)

- `ngIf` can specify a `then` clause
  - Lets you define the `<ng-template>` tag separately
  - Use `#` to give the `<ng-template>` tag a name

```
<div *ngIf="salary > 40000; then hiTaxTemplate1"></div>

<ng-template #hiTaxTemplate1> … </ng-template>
```

`ng-if-then-else.component.html`

If-Then-Else Tests (2 of 3)

- `ngIf` can also specify an `else` clause
  - Specify names of true/false `<ng-template>` tags

```
<div *ngIf="salary > 40000; then hiTaxTemplate2 else loTaxTemplate"></div>

<ng-template #hiTaxTemplate2> … </ng-template>

<ng-template #loTaxTemplate> … </ng-template>
```

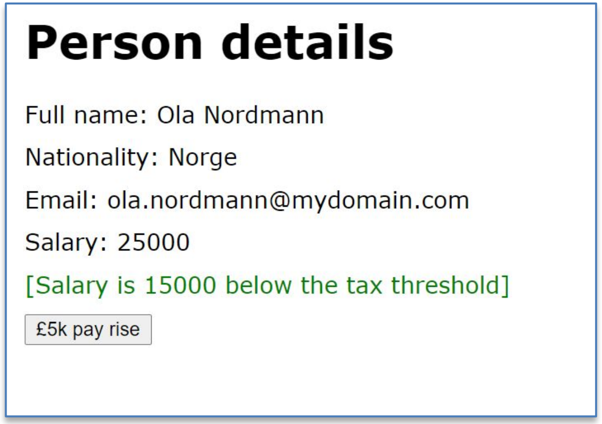
ng-if-then-else.component.html

Pearson

If-Then-Else Tests (3 of 3)

- Let's see an example of if-then-else tests
  - In the demo app, click the **ngIf-then-else** link

Initial display



Person details

Full name: Ola Nordmann

Nationality: Norge

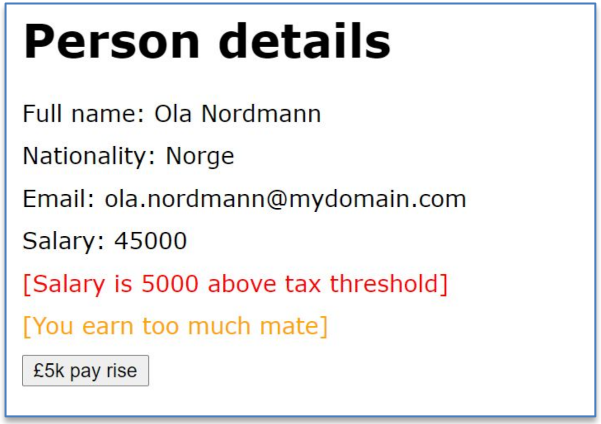
Email: ola.nordmann@mydomain.com

Salary: 25000

[Salary is 15000 below the tax threshold]

£5k pay rise

After several pay rises



Person details

Full name: Ola Nordmann

Nationality: Norge

Email: ola.nordmann@mydomain.com

Salary: 45000

[Salary is 5000 above tax threshold]

[You earn too much mate]

£5k pay rise

For Loops (1 of 2)

- You can use `ngFor` to iterate over a collection
  - Specify the collection you want to iterate over
  - Angular clones the element for each collection item

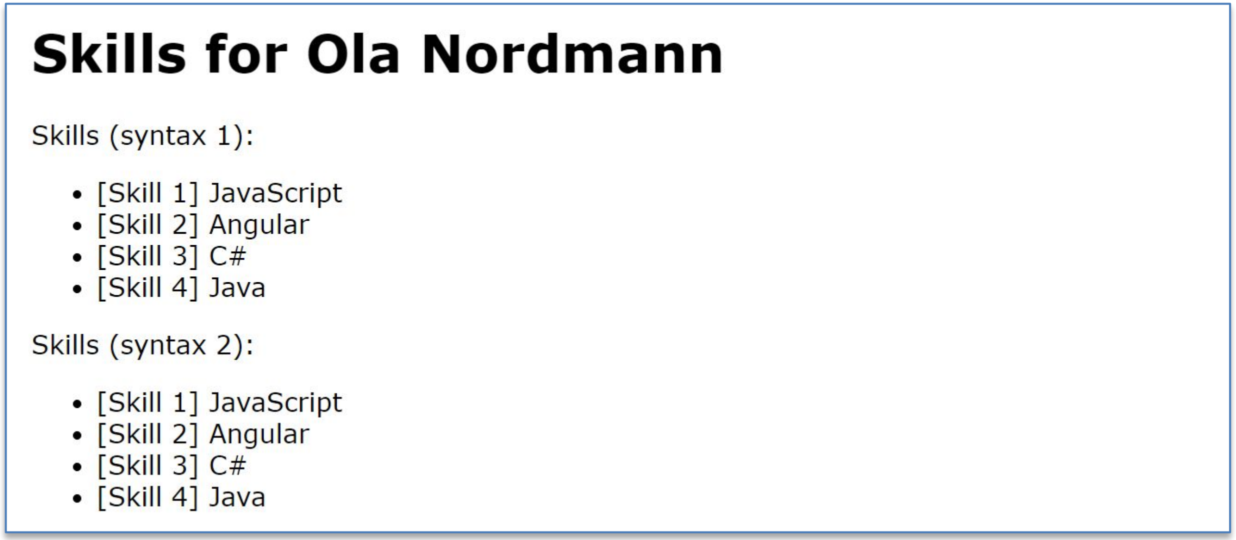
```
<ul>
  <li *ngFor="let s of skills; let i=index">
    [Skill {{i+1}}] {{s}}
  </li>
</ul>
```

`ngFor` exposes several useful variables:
- `index`, `first`, `last`, `even`, `odd`

`ng-for.component.ts`

For Loops (2 of 2)

- Let's see an example of `ngFor` loops
  - In the demo app, click the **ngFor** link



Skills for Ola Nordmann

Skills (syntax 1):

- [Skill 1] JavaScript
- [Skill 2] Angular
- [Skill 3] C#
- [Skill 4] Java

Skills (syntax 2):

- [Skill 1] JavaScript
- [Skill 2] Angular
- [Skill 3] C#
- [Skill 4] Java

Switches (1 of 2)

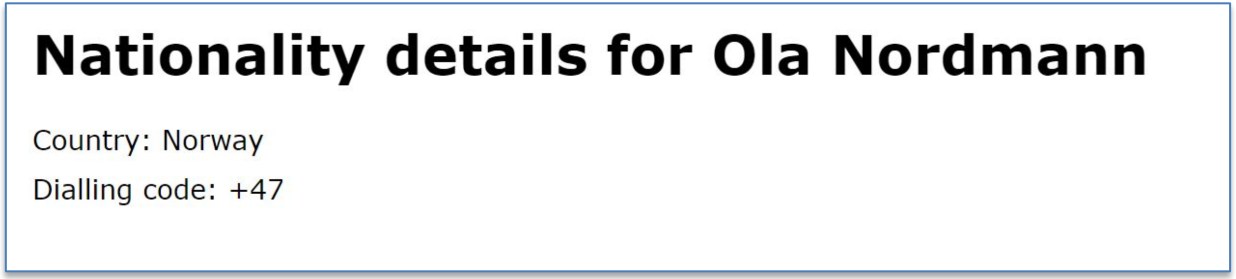
- You can use `ngSwitch` for multi-way branching
  - Use `ngSwitchCase` for each branch
  - Use `ngSwitchDefault` for default branch (optional)

```
<span [ngSwitch]="nationality">
  <span *ngSwitchCase="'Norge'">Norway</span>
  <span *ngSwitchCase="'UK'">UK</span>
  <span *ngSwitchDefault>[Other country]</span>
</span>
```

ng-switch.component.ts

Switches (2 of 2)

- Let's see an example of `ngSwitch` branching
  - In the demo app, click the **ngSwitch** link



Nationality details for Ola Nordmann

Country: Norway

Dialling code: +47

Summary

- Data binding in detail
- Two-way binding
- Decisions and iteration

Annex: Additional Techniques

- Data binding CSS classes and styles
- Key binding and template variables

Data Binding CSS Classes and Styles

- You can data bind a CSS class
  - Set `[class.aClass]` to a binding expression
  - If binding expression is truthy, CSS class is applied

```
<div [class.emphasis]="salary >= 40000">Salary {{salary}}</div>
```

- You can data bind a CSS style
  - Set `[style.aStyle]` to a binding expression
  - The binding expression specifies the style value

```
<div [style.color]="salary >= 40000 ? 'red': 'green'">Salary {{salary}}</div>
```

Key Binding and Template Variables

- Key binding
  - The `keyup` event ordinarily triggers on every keystroke
  - You can use the `keyup.enter` pseudo-event to target just the Enter keystroke

- Template variables
  - You can declare a template variable to represent an element, via the syntax `"#templateVarName"`
  - Allows you to access the element easily elsewhere

Key Binding and Template Variables

- In the demo app, click the **Additional techniques** link
  - Here's the relevant HTML and code

```html
<h1>Skills for {{name}}</h1>
<input #newSkill (keyup.enter)="addSkill(newSkill)">
<button (click)="addSkill(newSkill)">Add</button>
<p>{{skills}}</p>
```

additional-techniques.component.html

```typescript
export class AdditionalTechniquesComponent {
  name = "Kari Nordmann";
  skills = "";

  addSkill(newSkill: HTMLInputElement) {
    this.skills += newSkill.value + " ";
    newSkill.value = "";
    newSkill.focus();
  }
}
```

additional-techniques.component.ts

Pearson