Angular

Building the next version of the web with browser applications

CENTRIQ TRAINING

Prerequisites

- HTML / CSS
  - recommended course: 400 HTML-CSS
- JavaScript programming experience
  - recommended course: JavaScript
  - recommended course: JavaScript Tooling

Book

- Angular 5 Projects

Other books

- Advanced choices
  - **Essential Angular** by Victor Savkin (the router guy who left)
  - **Angular Router** by Victor Savkin

Exercises

- Completed exercises for the current version will be kept at
  - http://github.com/doughoff/wd-530v9

Intro to Angular

History

- 2009 – team started with Brad Green, manager
- Sep 2012 - 1.0.2
- **Sep 2016 – Angular 2**
- **Mar 2017 – Angular 4**
  - skipped 3.0, breaking changes due to router
- Nov 2017 – Angular 5
- May 2018 – Angular 6
- **Oct 2018 – Angular 7 - Angular Material**
- May 2019 – Angular 8
- **Feb 2020 – Angular 9 - Ivy**

Resources - official

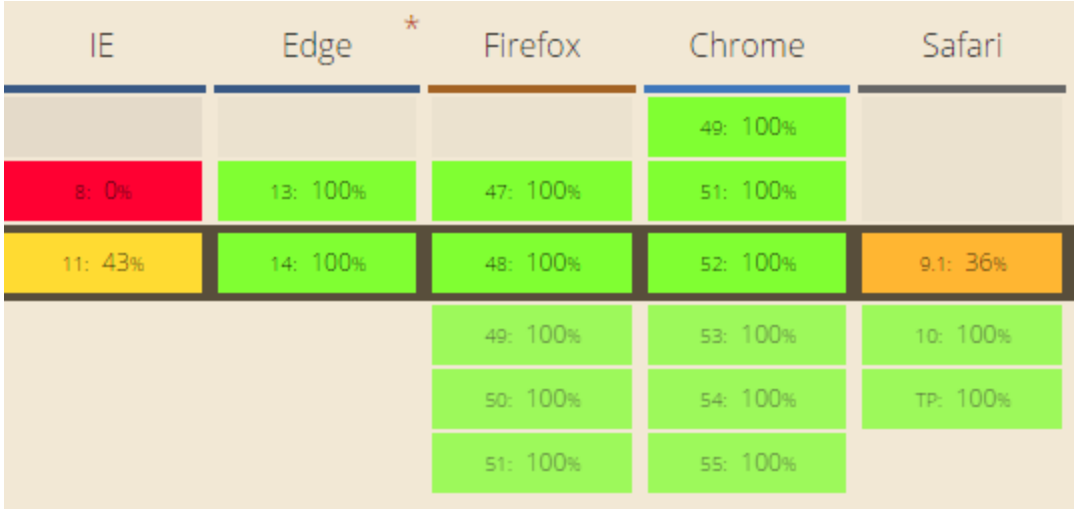
- Site: https://angular.io/
- Code: https://github.com/angular
- Docs: https://angular.io/docs/
  - Cheatsheet - https://angular.io/docs/ts/latest/guide/cheatsheet.html
- Blog: http://angularjs.blogspot.com/
- Milestone watch: https://github.com/angular/angular/milestones

Resources - minor

- Google Groups: http://ng-learn.org/
- Angular Modules: http://ngmodules.org/
- AngularJS 1 site: https://angularjs.org/
- Torgeir Helgevold articles - http://www.syntaxsuccess.com/angular-2-articles
- Design docs: https://drive.google.com/drive/u/0/folders/0B7Ovm8bUYiUDR29iSkEyMk5pVUk

Language support

- **ES5:** "today's JavaScript"
  - The easy, safe choice for Angular 1.
- **ES6:** ECMAScript 2015 or ES6
  - Partially support in current browsers, real applications require compiling.



Angular CLI

- https://cli.angular.io/
- Scaffolding tool
  - Based on Ember's CLI
- Automates basic tasks for setup and boilerplate code
- Installs
  - Instabul, Jasmine, Codelyzer, Karma, Protractor, tslint

Other

- Animations
  - https://angular.io/docs/ts/latest/guide/animations.html
- Testing with Jasmine, Karma, Augury (Chrome extension
  - https://angular.io/docs/ts/latest/guide/testing.html
  - https://augury.angular.io/
- RxJS ("Reactive Extensions")
  - asynchronous observable pattern, Microsoft project forked for any language
  - https://github.com/ReactiveX/RxJS

Hybrid apps using Angular

- Ionic Framework 5
  - https://ionicframework.com/
  - Build native apps from JS/TS APIs
- NativeScript
  - https://www.progress.com/nativescript
  - Build native apps with XML custom language

How to plan one-page apps

Planning an app

Architecture - SS framework

Server – ASP.NET or MVC

- Routing
- Controller logic
- Page generation
  - Data binding
  - Templates
- Security
- Services for client
  - data extraction

Client – jQuery, Bootstrap, etc.

- user triggered CSS
  - click / touch / hover
- user triggered server process
- browser triggered CSS
  - screen width

Architecture – SPA framework

Server – static files

- Services for client
  - **Security**
  - **Data**

Client – browser with

- **Routing**
- **Controller logic**
- **Page generation**
  - **Data binding**
  - **Templates**
- **Security**
- user triggered CSS
  - click / touch / hover
- user triggered server process
- browser triggered CSS
  - screen width

Angular features

- Page generation
  - Data binding
  - Templates
- Controller logic
- Routing
- Reusable components

SPA only

- Google assumes this design
- Most examples and tutorials target this design

Combined SPA & SS frameworks?

- Server side provides better security
- One side provides less distributed problems
- Client side operation can be extended with less complex packages

Possible SPA & SS framework designs

- Combine operations into an app on a SPA with the same model
  - One row – details, update, delete, duplicate
  - Multiple rows, same schema – browse, search, bulk data operations
  - Multiple rows, different schema – display, rearrange, insert, drop
- Create SS reusable view component library
  - Web Components

Best Practices

- Angular2 Styleguide
  - https://angular.io/styleguide
- Codelyzer
  - https://github.com/mgechev/codelyzer
  - for code reviews, linting, … soon static code analysis, template analysis, auto suggest
  - current: tslint
  - links to styleguide, live advice, in angular-cli
  - https://www.youtube.com/watch?v=bci-Z6nURgE&feature=youtu.be (May 2016) - Minko Gechev

Angular Material

- https://material.angular.io/
- UI library for fast building of mobile style apps

Other Material

- Material Design Lite – no framework
  - https://getmdl.io/
  - Angular Material vs Material Design Lite
    - https://scotch.io/bar-talk/angular-material-vs-material-design-lite
- AngularJS Material 1.1.1
  - https://material.angularjs.org/ for ng1

Setup

Setup choices – code + scaffold

- **Angular CLI
  - https://cli.angular.io/

TypeScript options

```
{
"extends": "../tsconfig.json",
"compilerOptions": {
"outDir": "../out-tsc/app",
"module": "es2015",
"types": []
},
"exclude": [
"src/test.ts",
"**/*.spec.ts"
]
}
```

TypeScript options

- emitDecoratorMetadata: true
  - transpiles necessary info for IDE – lots of errors if you don't!
- "noStrictGenericChecks": true
  - fixes rxjs 5.0 generic error with
  - or
  - "rxjs": "5.5.0", in Angular's package.json

Type definitions config

- https://github.com/typings/typings
- manage and install TypeScript definitions

```
{  "globalDependencies": {
"core-js": "registry:dt/core-js#0.0.0+20160725163759",
"jasmine": "registry:dt/jasmine#2.2.0+20160621224255",
"node": "registry:dt/node#6.0.0+20160831021119"
}
}
```

Typings

The TypeScript Definition Manager

Npm's inventory

```
"dependencies": {
    "@angular/animations": "~9.0.3",
    "@angular/common": "~9.0.3",
    "@angular/compiler": "~9.0.3",
    "@angular/core": "~9.0.3",
    "@angular/forms": "~9.0.3",
    "@angular/platform-browser": "~9.0.3",
    "@angular/platform-browser-dynamic": "~9.0.3",
    "@angular/router": "~9.0.3",
    "rxjs": "~6.5.4",
    "tslib": "^1.10.0",
    "zone.js": "~0.10.2"
},
```

RxJS

- install locally with npm rxjs
- Provides reactive programming syntax

- v6 additions:
  - "rxjs-compat": "^6.0.0",

<base href="/">

- Compatibility
  - IE10+
- Alternatives to base
  - Provide the router with an appropriate APP_BASE_HREF value.
  - Use absolute URLs for all web resources: css, images, scripts, and template html files.

<base href="/">

- If the application base changes you can use
  - <script>document.write('<base href="' + document.location + '" />');</script>
- This grabs the current URL
  - used in Google's documentation

<base href="/">

- Insert in <head> before any URL reference that might use it
- Sets a prefix to any relative URL path on the page
  - <base href="/">
  - <base href="/pages/baseball">
- Necessary to form html5 style URLs which use history.pushState
- can also use the target attribute to always open a new page

App selector

- <app-root>Loading...</app-root>
  - get Font Awesome link from cdnjs.com
- no inputs
- no outputs
- Only one selector, only one app.

/src/main.ts

The bootstrap

- allows for better testing
- platform specific
  - Cordova, Telerik NativeScript

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from
'@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from
'./environments/environment';
```

Dev vs. Prod

/src/main.ts
/environments/environments.ts

```
if (environment.production) {
enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppMod
ule)
.catch(err => console.log(err));
```

```
export const environment = {
production: false
};
```

/app/*.module.ts

Module configuration

- Introduced to provide Ahead-Of-Time compilation for faster precompile on server

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

/app/*.component.ts

Component classes

```
import { Component } from '@angular/core';
@Component({
  • selector: 'app-root',
  • // styles : `[p:color:red, div: color:green]`
  • // template:`<h1>An AngularJS 2 App</h1>
  •                 <p>… </p>`,
  • styleUrls: ['app.component.css'],
  • templateUrl : `app.component.html`
})
export class AppComponent { }
```

moduleId – v4

- v.4.0 (pre-March 2017)
- Component relative (webpack problem)
  - @Component({
  - **moduleId: module.id,**
  - templateUrl : `./basic.component.html`,
  - styleUrls: [./basic.component.css']
- Remove any moduleId now

Modules

- ES6 / TypeScript
  - not required by Angular but very recommended

- barrels – collections of modules
- bundle – a file for all the code of one or more barrels

Modules – import & export

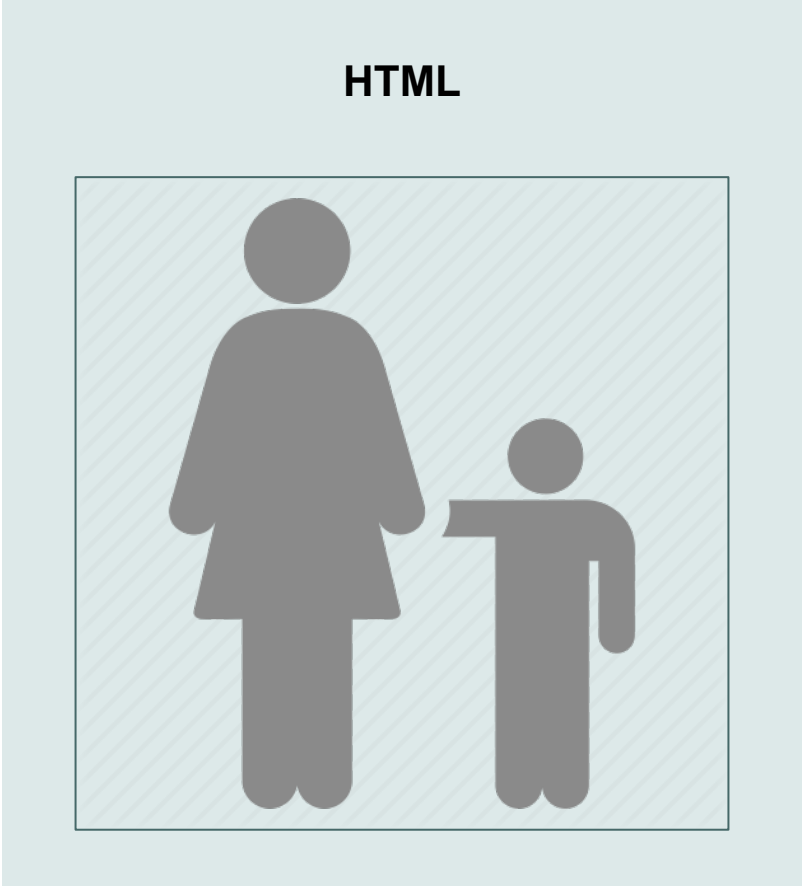
- **import** { Component } from “**@angular/core**";
  - allow use of class Component from a .js file called core
- **export** class HelloName { }
  - allow use of class HelloName by another import
  - functions and values can be exported also
- import and export use ES6 module syntax
  - http://www.2ality.com/2014/09/es6-modules-final.html

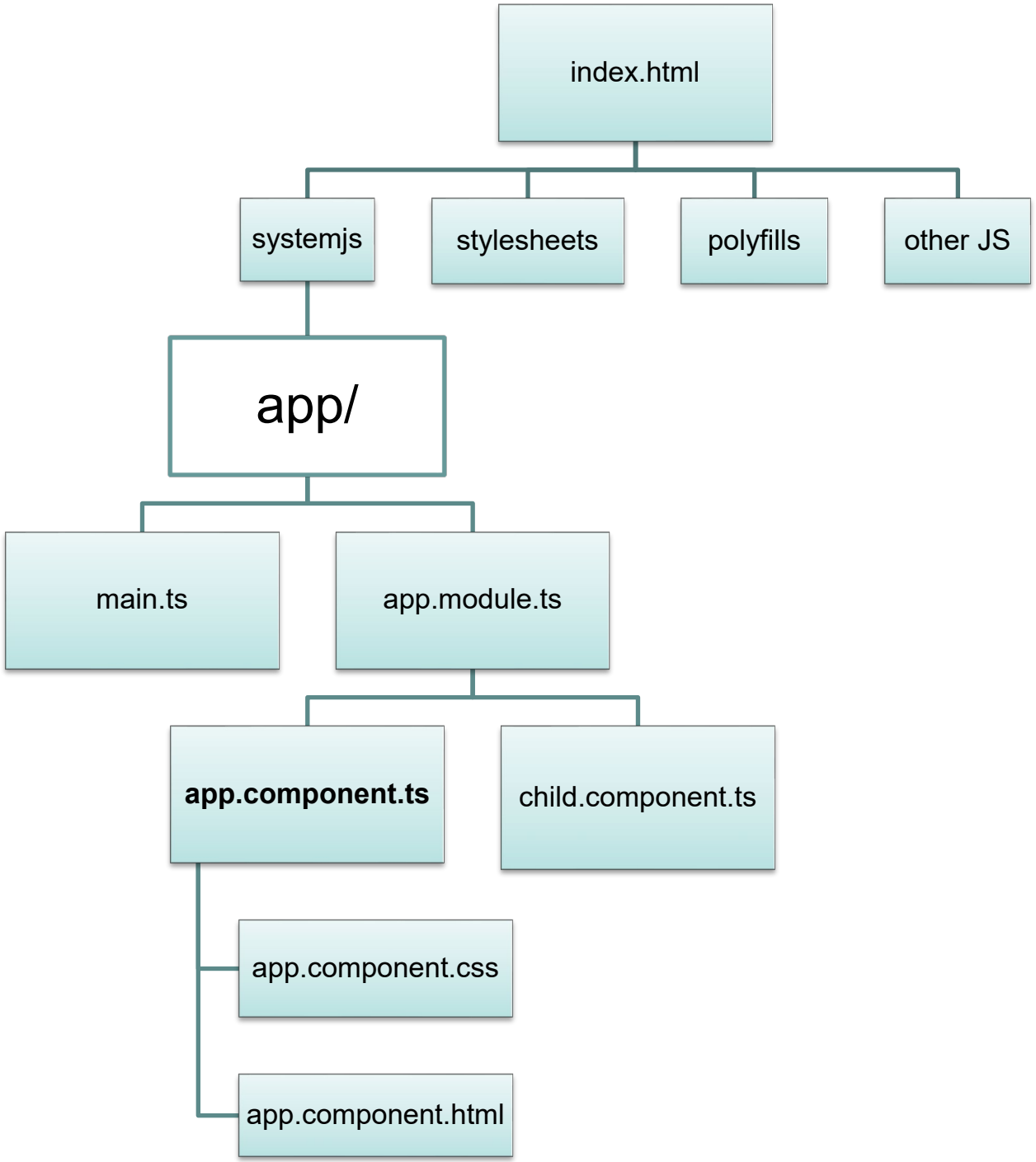
Component

- The area of the DOM that you want to manage
  - the view scope
- Three parts
  - metadata configures the code
    - defines what tag to use
    - uses TypeScript's decorators: @Component
  - a template defines the HTML and data variables
    - uses {{ mustache tags }}
    - uses special attributes
  - a class defines the view logic and data

App structure

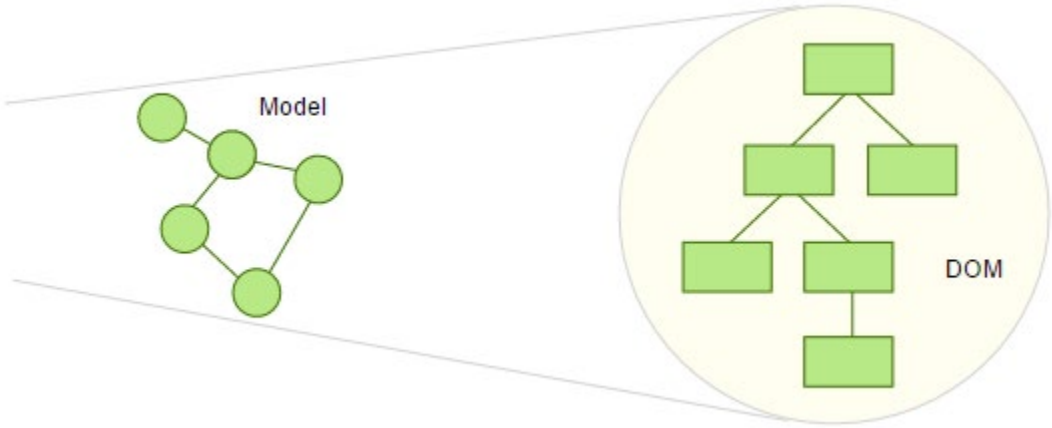
- HTML page
  - polyfills for older browsers
  - HTML
    - root app selector
      - component code
      - template
      - styles
      - dependencies
      - child components selectors
        - child component code…




index.html
systemjs
stylesheets
polyfills
other JS
app/
main.ts
app.module.ts
app.component.ts
child.component.ts
app.component.css
app.component.html

Data binding – projecting data

- Data model in code → DOM
- mapping, projecting, no change
- updates require mapping/binding
- must track state (the model/DOM data)



Data binding – app.component.ts

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

@Component({
    moduleId: module.id,
    selector: 'hello-name'
    template:`<div>Hello, {{name}} </div>`
})
export class HelloName {
    private name: string = 'world';
}
```

Templates – inline vs file

- **template**:`<div>Hello, {{name}}</div>`
  using ES6 template strings

  or
- **templateUrl**: ‘./hello_name.html'
  - path from component

Component view

```
class HelloNameApp {
      private name: string;
      constructor( ) {
            this.name = 'world';
      }
}
```

Development to production

- You will see on the console:
  - Angular 2 is running in the development mode. Call enableProdMode() to enable the production mode.
- To make faster for production

```
import { NgModule, enableProdMode }  from
'@angular/core';
enableProdMode();
```

Exercises

- 6 Setup using CLI project with Code
- 7 Load CLI project with local server extension
- 8 Measure template's resource loading times

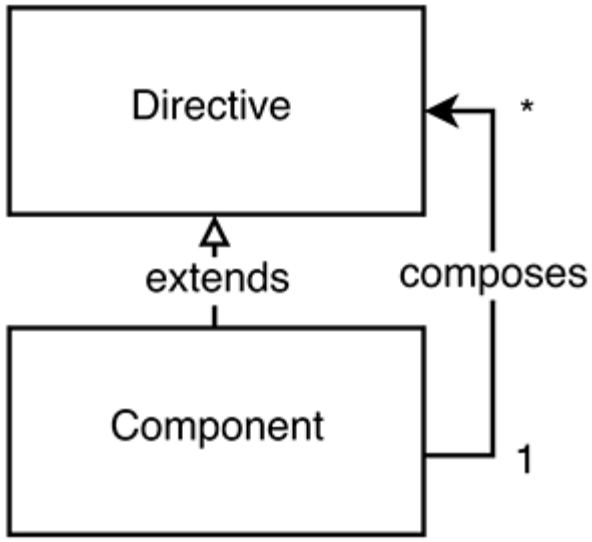
Components

Directive

- Three types
  - **Component** – main unit of Angular
  - **Structural** – many built-in logic functions for layout
    - < employee ***ngIf**="isEmployed"></employee >
  - **Attribute** – alters behavior or appearance by adding attribute syntax
    - <input **[(ngModel)]** ="employee.name">

Components

- Directive - Holds logic, but no structure, base class
- Component - Extends a Directive and is composed of other directives or components.



Selectors

- element, class, or attribute syntax
- selector: 'custom-box, .custom-box, [custom-box]', :not( )
  - **<custom-box>Matching tags</custom-box >**
    - < custom-box /> is not valid – must use a closing tag, not empty
  - <span **class= 'custom-box'** >a class
  - <span **custom-box** >an attribute
- selector: '.custom-box:not(h1)'

Selectors

- Not valid:
  - ids, ancestor/child, ng1 comments, ng1 alternate naming syntax (custom:box)
- Recommended – kebab-case
  - meat-onion-tomato-meat-onion-meat
- Not as recommended
  - camelCase / PascalCase

Module declarations

- needed for initial component & child components
- @NgModule ({imports […],
  declarations [Person, Beach]
- ----------
- @Component({ selector: '**person**', template: `.o.`}) class Person{ }
- @Component({ selector: 'beach', template: `
  The Beach: <person></person>
  <person></person>~~~~`}) class Beach{ }

Styles

Styles - internal

- Defined in the @Component decorator
- Written to a style element in the rendered page.
- Styles are bounded by the element of the selector (view encapsulation)
  - Emulated View Encapsulation - default
- **styles:** ['.primary {color: red}', '…']
  - an array of rule-sets, not a multi-line string for all!
- Webpack and other module bundlers
  - styles: [require('my.component.css')]

Styles - external

- **styleUrls:** ['./my-component.css', '…']
  - uses relative references when using

Styles – in template

- Use a <style> element at the top of your template to replace the **styles** or **styleURLs** of the Component decorator

Style strategy

- styles: - poor tool support
- styleURLs
  - **No transpile necessary!**
  - **Tool support**
  - good for very large libraries
  - Access by designers
  - Uses base href, start relative URL without slash!
- template <style> - easily read, updated, managed

Style scoping

- **Emulated** - adds attribute to scope to component - default
- Native – uses browser's shadow DOM
- None – no scoping, styles are cross boundary from component to DOM, ~global
  - @Component {
    encapsulation: ViewEncapsulation.None
    … }

Styles – special selectors

- **:host** { display: block; border: 1px solid black; }
  - Applies to containing component
- **:host(.active)** { border-width: 3px; }
  - Applies to containing component only when it has active class
- **:host-context(.theme-light) h2** { background-color: #eef; }
  - Applies to containing class child H2 elements if some ancestor has theme-light class

Styles - special selectors

- :host **/deep/** h3 { font-style: italic; }
  - Forces (releases encapsulation for) style so any H3 descendent of containing component is styled
  - Only for emulated
- :host **>>>** h3 { font-style: italic; }
  - Alternate syntax for above

Exercises

- 9 Create a template
- 10 Use different selectors
- 11 Add more style

Templates

Templates - inline

- Inline template
  - can use ES6 backticked text (template literals)
  - **template:** `
    - <div *ngFor=“let talk of talks"> {{talk.title}} by {{talk.speaker}}: {{talk.description}}
  - `

Templates - external

- External template - best
  - @Component({
    - **templateUrl: 'template.html'**

Syntax

- All HTML is valid except
  - <script> - to prevent injection attacks
  - <html>, <body>, <base>

Syntax - literals

- Text – quoted literals or expression
  - {{ 'Hello' }}
  - {{ 1 + 11 + 111 }}
- Text concatenation
  - {{ 'Hello' + ', world' }}
- Text interpolation
  - And he said "{{ 'Hello!' }}" to the world
- With a text filter
  - {{ 'Hello' + ', world' | uppercase }}

Syntax - literals

- Alternative syntax for {{ }}
  - Hello {{name}}
  - <div [textContent]="interpolate(['Hello'], [name])">

Syntax - literals

- Mixed with ASP.NET server data bindings
  - {{ '<%= DateTime.Now %>' }}
  - server code executes first, then renders client side literal
- Mixed with attribute value text
  - <a href="img/{{ username }}.jpg">

Expressions

- {{ the Angular expression }} can be
  - {{ any @Component class member private field }}
    - {{ totalItems + 'items' }}
  - {{ any @Component class member method }}
    - {{ getQuantity }}, {{ calcQuantity( ) }}
- result can be assigned to an element or directive property
- best practice
  - use data properties and methods to return values and no more

Elvis /safe navigation operator ?.

- guards against null and undefined values in property paths
  - view will disappear on null parent object
- Employer: {{employer?.companyName}}
  - if employer field is optional and undefined or null, the rest of the expression is ignored.
- Can be swapped out with longer version
  - Employer: {{employer && employer.companyName}}
- C# null coalescing operator 6.0

Ternary operator ? yes : no

- An expression to replace num > value ? 50 : 20

```
{{
   • {true: 50, false: 20}[num > value]
}}
```

Not used

- Prohibited
  - Assignment except in Event Bindings.
  - **new** operator
- Not supported
  - bit-wise operators, | and &
  - ++, --
  - access to global namespace, window, or document
    - console.log( )

Binding

Data binding – one way values

- interpolation
- from class (data source) to template in DOM (view target)
- most often a @Component class property
- template:`
  - `<input type="text" value="{{name}}" />`
  - `<div>Hello, {{name}}!</div>`
- export class BuiltIn {
- private **name**: string = 'John Smith';

}

HTML attribute binding

- HTML attributes
  - Includes global HTML attributes – class, id, style, title, etc.
    - https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes
  - Includes specific element attributes
    - https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input
    - https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement
  - Binding example
    - <input type='text' class='{{myClassName}}' id='{{idNumber}}' value='{{defaultValue}}' {{hasFocus}}>

DOM property binding

- DOM properties using JS code
  - Includes Element properties
    - https://developer.mozilla.org/en-US/docs/Web/API/Element
  - Includes specific properties
    - https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement
  - Binding example
    - <input type='text [value]='defaultValue'>
    - vs. <input type='text‘ value='{{defaultValue}}'>

DOM property binding

- set the default value of an input from the class
- template:`
- <input **[value]** = "**defaultName**"> `
- })
- export class BuiltIn {
- private **defaultName** : string = 'John Smith';

}

  - value does not work with ngControl="…"

DOM property binding

- standard syntax
  - <input **[value]** = "**defaultName**">
- alternate syntax / canonical form
  - **<**input **bind-**value = "defaultName">

DOM property binding

- <button [disabled]="isUnchanged"> Save </button>
- The property of the DOM element
  - or Component or Directive
- Attributes initialize DOM properties – final
  - watch the DevTools when you update a text field
- DOM property bindings are not final
  - button disabled="false" does not work
  - button [disabled]="isInvalid()" does work

DOM property binding

- HTML attributes that are also properties will be converted so either syntax is OK
  - <input type="text" value='{{myName}}' />
  - <input type="text" [value]='myName' />

  - <img src="{{heroImageUrl}}">
  - <img [src]="heroImageUrl">

  - The title is {{title}}
  - <div [textContent]="'The title is '+title">
  - <div [innerHTML] ='text' ></div>

HTML attribute vs DOM property binding

- Some HTML attributes are DOM properties
- Some HTML attributes don't have corresponding DOM properties.
  - colspan
- Some DOM properties don't have corresponding HTML attributes
  - textContent
- Many HTML attributes appear to map to properties ... but not the way we think!

Class property binding

- CSS Class
  - .myClassName { }
  - <div [class.myClassName]="isTruthy">
    - isTruthy = true
  - <div [class]=“myClassNameVariable">
    - myClassNameVariable ='myClassName'

Style property binding

```
<button [style.color] = "isSpecial ? 'red' :
'green'">

<div [style.background-color] = 'background'
                [style.color] = 'foreground'>
</div>
```

```
export class BuiltIn {
  private background: string = 'hsl(200,80%,90%)';
  private foreground: string = 'hsl(200,80%,40%)';
}
```

Style property binding – unit selection

- units are not a property in JavaScript

```
[style.font-size.px]="fontSize"
[style.font-size.em]="fontSize"
[style.font-size.%] ="fontSize"

<div [ngStyle]="{'font-size': fontSize+'px'}”>
```

Style property binding - ngStyle

- bulk style mapping

```
<div [ngStyle]="setStyles()">
  This div is italic, normal weight, and x-large
</div>
```

```
setStyles( ) {
  return {
    'font-style': this.canSave ? 'italic': 'normal',
    'font-weight': !this.isUnchanged ? 'bold':'normal',
    'font-size': this.isSpecial ? 'x-large': 'smaller'
  }
}
```

Class property binding - DOM

- add or remove CSS class names
  - appends to class attribute
- <div **[class]**="myClassName">
  - appends the class of value of myClassName
- <div **[class.myClassName]**="isTruthy">
  - appends class property if value is truthy
    - Not truthy values are 0, no text, and false

      Also undefined and null in JS

Class property binding - ngClass

- bulk class assignment
- a map of classes to append with their corresponding boolean tests

```
<div [ngClass]="{
   active: isActive,
   disabled: isDisabled,
   'has-error': hasErrors
}">
```

Class property binding comparison

- <input class='hide bold'>
- [class.hide]='whenValidFor.first'
- [class.bold]='whenBoldFor.first'
  - reads better for a few classes
- [ngClass]='{
  hide : whenValidFor.first ,
  bold : whenBoldFor.first
  }'
  - better for many classes

Attribute property binding

- exception to no attribute changes
  - useful when no property exists
- <button [**attr.**aria-label] = "help">help</button>
- <div **[attr.**role**]** = "myAriaRole">
- <tr><td [**attr**.colspan] = "{{1 + 1}}">Three-Four</td></tr>

Other property bindings

- Directive property (input)
  - <div [ngClass] = "{selected: isSelected}">
- Component property
  - <hero-detail [fromParentComponent]="currentHero"></hero-detail>

Template #ref variables

- Binds name to node for later use

```
<img src="dog.jpg" alt="This is my dog." #dogPic>
<figcaption>
   {{'Caption' + dogPic.alt}}
</figcaption>
```

NgNonBindable

- to prevent Angular from processing template

```
<div class='ngNonBindableDemo'>
<span class="bordered">{{ content }}</span>
<span class="pre" ngNonBindable>
  • This is what {{ content }} rendered
</span>
</div>
```

Exercises

- 12 Get text from component
- 13 Set attributes from component
- 14 Use a dog-panel model class

View logic

Structural directives

- *ngFor
- *ngIf

- Uses asterisk for sugar syntax to produce a **<template> element** in shadow DOM of element that has the attribute

*ngFor - syntax

- iterate over a collection from Component class
- declare local variable in template of each item

```
<ul>
    <li *ngFor = "let item of names">
        Hello, {{ item }}!
    </li>
</ul>
```

*ngFor – implicit values

- available values
  - **index** : int
  - first : boolean
  - last : boolean
  - even : boolean
  - odd : boolean

```
<div *ngFor="let city of cities; let i = index">
   #{{i+1}}: {{city}}
</div>
```

*ngIf - syntax

```
@Component({
    selector: 'built-in',
    template:`
    <div *ngIf ="x > y">
        x bigger than y
    </div>
    <div *ngIf ="x <= y">
        x less than or = to  y
    </div>`
})
export class BuiltIn {
    private x : number = 300;
    private y : number = 200;
}
```

switch – syntax for values

```
<div [ngSwitch]="x>y">
   <span *ngSwitchCase="true" > x > y </span>
   <span *ngSwitchCase="false"> y >= x </span>
   <span *ngSwitchDefault> default text </span>
</div>

<div [ngSwitch]="x>y"><span>
   <ng-template [ngSwitchCase]="true">x > y</ng-
template>
   <ng-template [ngSwitchCase]="false">y >= x</ng-
template>
   <ng-template ngSwitchDefault >default text</ng-
template></span>
</div>
```

switch – syntax for literal strings

```
<div [ngSwitch]="stringVar">
  <ng-template ngSwitchCase ="a">aaaa</ng-
template>
  <ng-template ngSwitchCase ="b">bbbb</ng-
template>
  <ng-template ngSwitchDefault >not a or b</ng-
template>
</div>
```

Exercises

- 15 For directive
- 16 If directive
- 17 Switch directive

Pipes

Intro

- serves same purpose as a custom get method
- uses a transform function to alter values on the View
- called filters in ng1

Pipe operator |, parameter :

- Single
  - {{ title | lowercase }}
- Chained
  - {{ birthday | date:' ' | uppercase }}
- Configured
  - Birthdate: {{currentHero?.birthdate | date:'longDate'}}

Pipes - common and custom

- Common
  - The hero's birthday is {{ birthday | date:' ' }}
- Custom
  - Card No.: {{ cardNumber | myCreditCardNumberFormatter }}

Common pipes – date

- uses the Internationalization API – IE11+, no Safari
- will not re-evaluate
- expression | date : format
  - expression is Date object or # of ms since UTC
  - format – check table at https://angular.io/docs/ts/latest/api/common/index/DatePipe-pipe.html

Common pipes – date

- date : 'MMMMdy' or
  date : 'longDate' = September 3, 2010
- date: 'yMd' or
  date: 'shortDate' = 9/3/2010
- date: 'jm' or
  date: 'shortTime' =12:05 PM

Common pipes - currency

- uses the Internationalization API – IE11+, no Safari
  - https://en.wikipedia.org/wiki/ISO_4217
- expression | currency : <currency code> : <digit info> : <symbol display>
  - digit info: see decimal pipe
  - symbol display: ‘code’ (USD), ‘symbol’ ($), or symbol-narrow ($)
- amount | currency:'USD’:’code’
- amount | currency:'EUR,’symbol','4.2-2’'

Common pipes - decimal

- expression | number: <digit info>
  - digit info: <minIntegerDigits | 1>.<minFractionDigits | 0> -<maxFractionDigits | 3>

```
{{ e | number:'3.1-5' }}
{{ pi | number:'3.5-5' }}
```

Common pipes - percent

- expression | percent : digitInfo
  - digit info: see decimal pipe

```
{{amount | percent:'4.3-5'}}
```

Common pipes – uppercase, lowercase

```
{{value | lowercase}}
{{value | uppercase}}
-----------------------------------------------------
template: `<p> {{ 'abc' | textCasingStyle }}</p>
<button (click)='toggleFormat( )'>Toggle Case</button>`

export class TestComponent {
   toggle = true;
   get textCasingStyle ( ) { return this.toggle ?
'uppercase' : 'lowercase'}
   toggleFormat( ) { this.toggle = !this.toggle; }
}
```

Common pipes - json

- Output
  - { "firstName": "Hercules", "lastName": "Son of Zeus",
  - "birthdate": "1970-02-25T08:00:00.000Z",
  - "url": "http://www.imdb.com/title/tt0065832/",
  - "rate": 325, "id": 1 }

```
<div>{{currentHero | json}}</div>
```

Common pipes - slice

- expression | slice : start : end
- positive start, up to but not including end
  - ['a', 'b', 'c', 'd'] | slice:1 : 3 → ['b', 'c']
  - 'abcd' | slice: 1: 3 → ['b', 'c']
- negative start from end, not including how many from end
  - 'abcdefghij' | slice: -4 → 'ghij'
  - 'abcdefghij' | slice: -4 : -1 → 'ghi'

Common pipes - async

- subscribes to an Observable, Promise or EventEmitter and returns the latest value it has emitted. When a new value is emitted, the async pipe marks the component to be checked for changes.
- the only common **stateful** pipe

Common pipes - async

```
@Component({
  selector: 'hero-message',
  template: 'Message: {{delayedMessage | async}}',
})
export class HeroAsyncMessageComponent {
  delayedMessage:Promise<string> = new
Promise((resolve, reject) => {
  setTimeout( () => resolve('You are my Hero!'),500 );
  });
}
```

Custom pipes – code pipe

- @Pipe decorator on class, transform( )

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

@Pipe({name: 'yourPipeName'})
export class YourPipeClass implements PipeTransform
{
  transform(value:string, args:string[ ]) : any {
    return 'a transformed value';
  }
}
```

Custom pipes – declare in module

- No need to declare in component since module covers that

```
import {CurlyQuotesPipe}  from './curlyquotes.pipe';
@NgModule({
    declarations: [ DogPanel, CurlyQuotesPipe,
DogDetail ],
```

Custom pipes - execute

```
@Component({
  selector: 'aTag',
  template: `{{'no change' | yourPipeName }}`,
})
export class PipeTest { }
```

Comparison

- Pipes are generally more readable but JavaScript can be used in place of them.

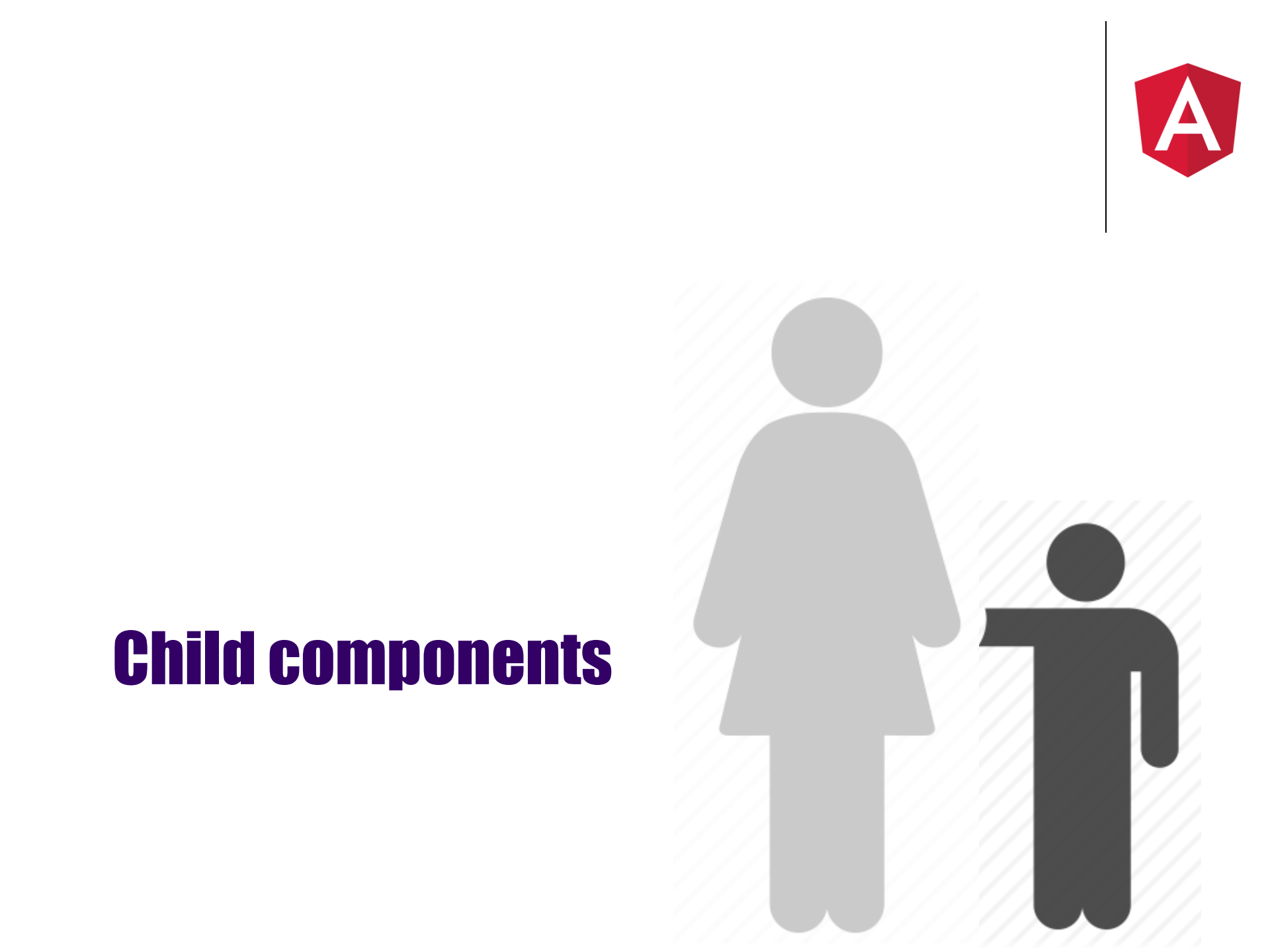
```
@Pipe({name: 'trim'})
export class TrimPipe {
  transform(value: string, args: any[]) {
value.trim(); } }
-----------------------------------------------
{{  name | uppercase | trim }}
{{ (name | uppercase).trim() }}
{{  name.toUpperCase().trim() }}
```

ng1

- number, orderBy, and filter are no longer used
- async, decimal, and percent are new to ng2

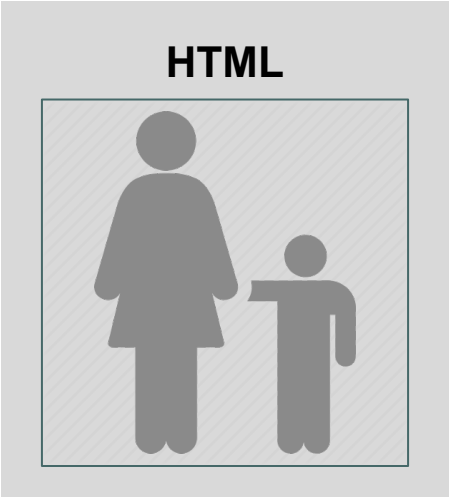
Exercises

- 18 Common pipes
- 19 Async pipe
- 20 Custom pipe


Child components

Intro

- Components can talk to each other
  - p2c - flow **data** from **parent to child**
  - c2p – flow **events** from **child to parent**
- Html page with app / root component
  - Can not see/compile data for root component
  - Security/architectural restriction
- Types of data
  - p2c - innerHTML, attributes, #vars
  - c2p – events, #vars

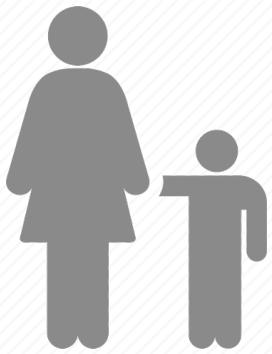


Smart & dumb components

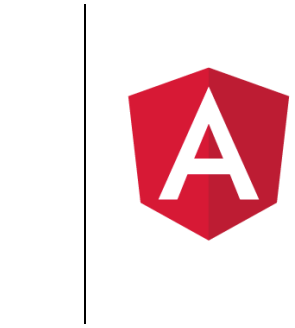
- Dumb / presentational component
  - Accepts data via inputs
  - Emits data changes via event outputs
  - stateless
- Smart / container component
  - Communicates with services
  - Renders child components
  - stateful

p2c – declaring child components

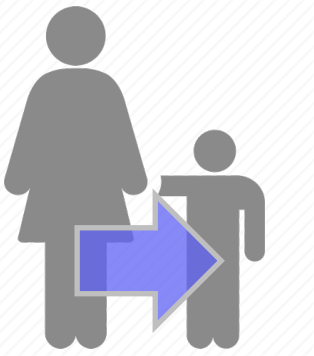
- use an import
  - import {Child} from './components/child';
- include a reference from module config to child
  - @NgModule({
    - imports: [BrowserModule],
    - declarations: [ParentComponent, **ChildComponent**, …]



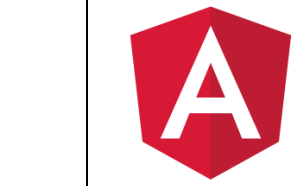
p2c – content projection



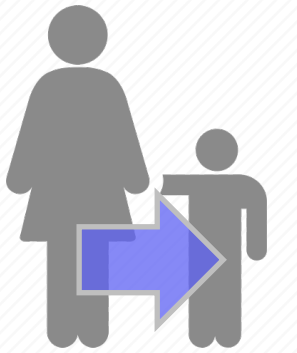
- Moves parent template's child element's innerHTML
- Child template 'queries' parent's innerHTML with <ng-content> element
- <ng-content selector='…'>
  - Collects all content matching selector
  - id attribute not implemented
- No selector
  - Gets all content not already selected
- ng1 - transclusion



p2c – content projection

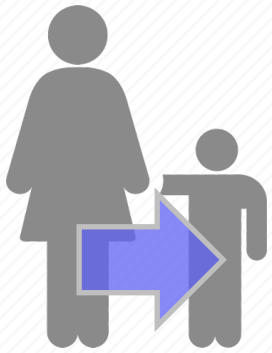


- **Parent template defines** elements & data
  - <child><stuff>This is elemental stuff 1.</stuff>
  - <div a>aaaaaaaaaaaaaattribute. </child>
- **Child template uses** for final data position
  - <ng-content select="stuff"></ng-content>
  - <ng-content select=".togetherness"></ng-content>
  - <ng-content select="[a]"></ng-content>
  - <ng-content select="planet[x]"></ng-content>
  - <ng-content></ng-content>

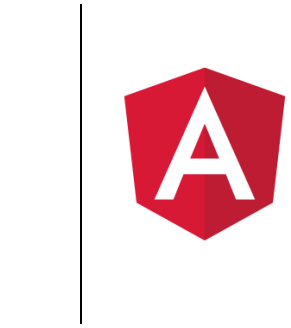


p2c – @Input - preferred

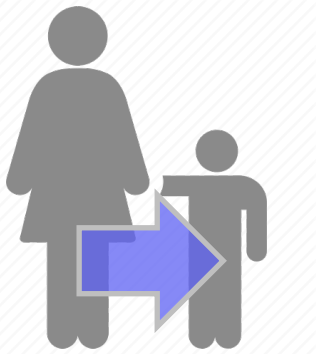
- **Parent template exposes** data
  - <child-component [**childVariableIn**]='childArgument' >
  - <child-component **childTextIn**='child text' >
- **Child component defines** interface fields
  - import { Input } from '@angular/core';
  - export class ChildComponent {
    - @Input( ) **childVariableIn** : string;
    - @Input('alias') **childTextIn** : string;
- **Child uses** fields
  - template - {{**childVariableIn**}} {{**alias**}}
  - code – **childVariableIn, alias**



p2c – inputs: [ ]

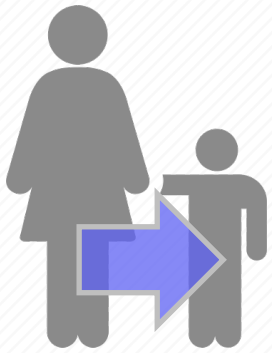


- **Parent template exposes** data
  - <child-component [**childVariableIn**]='childArgument' >
  - <child-component **childTextIn**='child text' >
- **Child component defines** interface fields
  - @Component({
    - inputs: ['**childVariableIn**']
    - inputs: ['**childTextIn** : alias']
- **Child uses** fields
  - template - {{**childVariableIn**}} {{**alias**}}
  - code – **childVariableIn, alias**



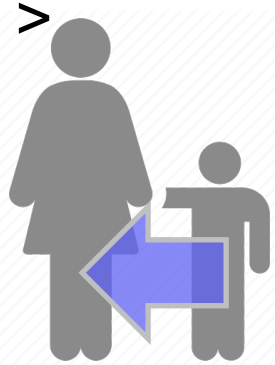
p2c – @Input property setter

- **Parent template exposes** data
  - <child-component [**childVariableIn**]='childArgument' >
- **Child component defines** interface field as setter
  - import { Input } from '@angular/core';
  - export class ChildComponent {
    - private _prop : string;
    - @Input( )
      set prop (**childVariableIn** : string) { this._prop = **childVariableIn** || 'zilch';}
    - get prop() { return this._prop; }
- **Child template/component uses** property – {{prop}}



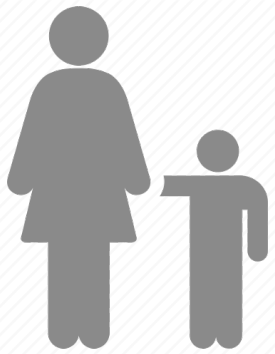
c2p – @Output

- Child component declares EventEmitter
  - import { Component, EventEmitter, Output } from '@angular/core';
  - export class ChildComponent {
    - @Output('alias') **emitter** = new EventEmitter<any>();
- Child component emits event
  - this.**emitter**.emit(payloadOut);
- Parent template exposes interface in child element
  - <child-component (**alias**)='onEvent(payloadIn)' >
- Parent component handles event
  - onEvent(data : any) { }



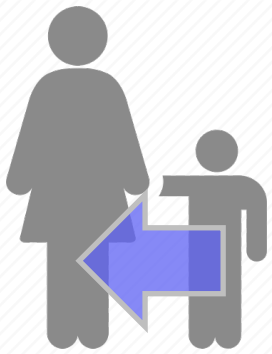
p2p – local variables

- uses pound sign before a scoped variable name for DOM element
  - also called a resolve
- <div **#newDiv** />
  - almost like **id='newDiv'** for cross element access
  - variable is now accessible from this element or in any descendant
  - alternative syntax <div **var-newDiv** />



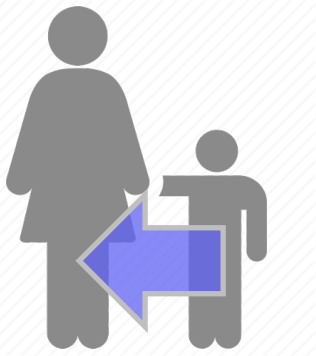
c2p – local child component var

- Child declares members
  - private field : any;
  - private function( ) : any { return 0; }
- Parent exposes child element
  - <child-component **#child** >
- Parent uses child's members in template
  - {{ **child**.field }}
  - {{ **child**.function( ) }}



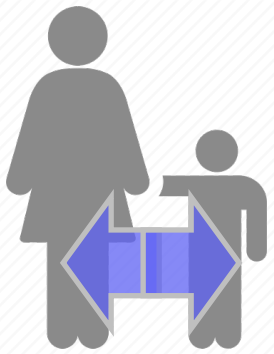
c2p – @ViewChild

- @ViewChild, @ViewChildren
  - @ViewChild(AChildComponent) appears first in class declaration
  - reference child elements inside parent template – shadow DOM
  - ViewChildren is a QueryList – Iterable, Observable
    - **first**, **last** is one
    - **changes** will alert you when it changes
- use child component methods
  - EventEmitter for child → parent methods



p2c/c2p – via service

- See Cookbook / Component Interaction / Parent and children communicate via a service
  - Message broker pattern
  - https://angular.io/docs/ts/latest/cookbook/component-communication.html#!#bidirectional-service



Summary

- Content projection
  - Uses child's ng-content select to associate to any children of parent's child selector
- P2C
- C2P
  - Declare child element in parent template with # to direct access fields and methods

Exercises

- 21 Content projection
- 22 Data input from parent
- 23 Event input from parent