AN INTRODUCTION TO...

ANGULAR 2

JOHN.GORTER@INFOSUPPORT.COM
TWITTER: @JOHNGORTER

AGENDA

- INTRODUCTION
- TYPESCRIPT
- COMPONENTS
- PIPES
- SERVICES
- FORMS
- ROUTING

CHAPTER

INTRODUCTION

INTRODUCTION

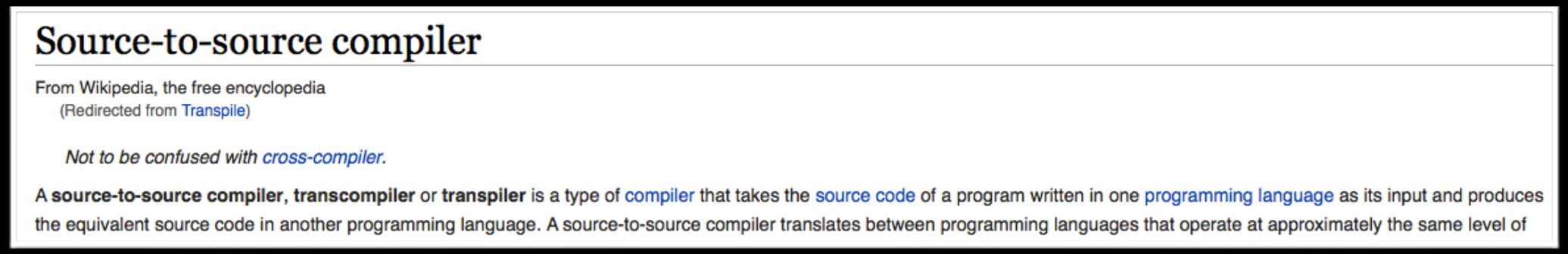
BENEFITS

- Structure (Everything of Angular 1)
- Speed
- Simplicity
- DOM Goodness

INTRODUCTION

DEVELOPMENT OPTIONS

- Without transpiling
  - Just deploy ES5
- With Build-time or Run-Time transpiling
  - Write ES6+ but run ES5
- Transpilers
  - Babel, Traceur,Typescript



Source-to-source compiler

From Wikipedia, the free encyclopedia
(Redirected from Transpile)

Not to be confused with cross-compiler.

A **source-to-source compiler**, **transcompiler** or **transpiler** is a type of compiler that takes the source code of a program written in one programming language as its input and produces the equivalent source code in another programming language. A source-to-source compiler translates between programming languages that operate at approximately the same level of

INTRODUCTION

DEVELOPMENT OPTIONS

- What is ES6?
  - ES6 lets you write classes and modules without the complex syntax of ES5
- ES6 most important features include:

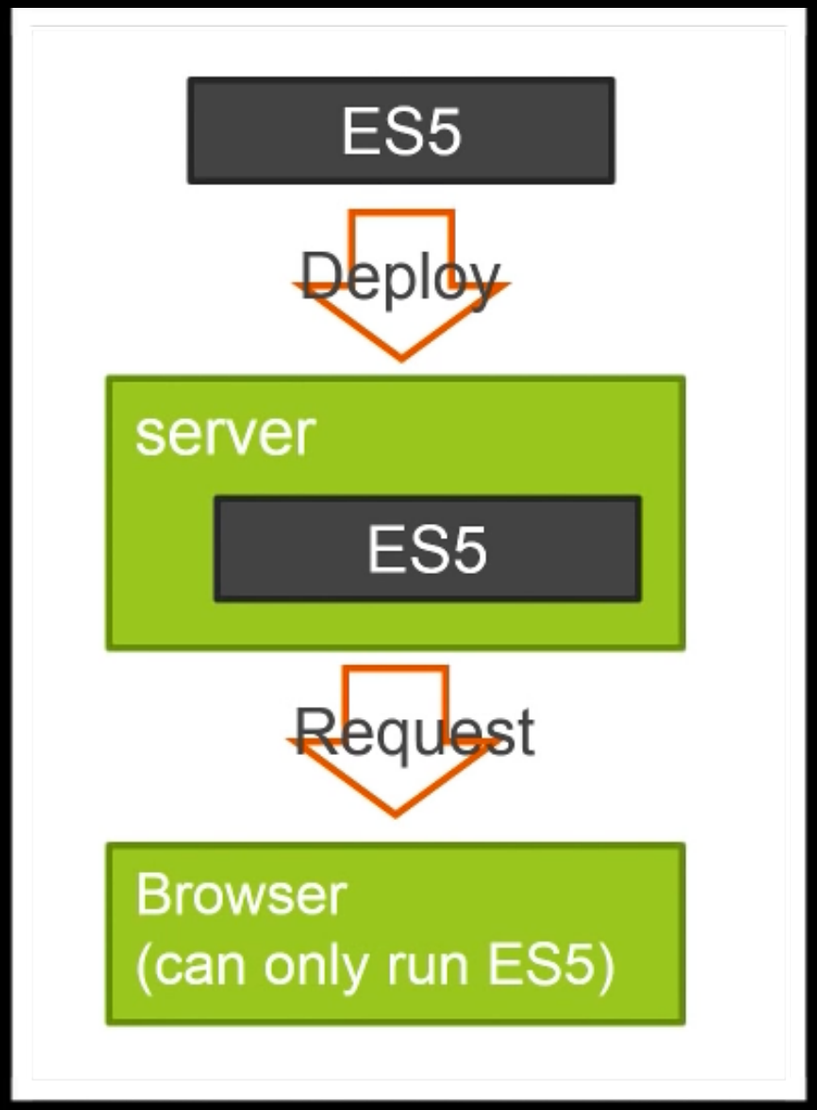
  arrows, classes, enhanced object literals, template strings, destructuring, default/rest /spread, let/const, iterators + for..of, generators, unicode, modules, module loaders, map/set/weakmap/weakset, proxies, symbols, subclassable built-ins, promises, math/number/string/array/object APIs, reflect api, tail calls

  https://github.com/lukehoban/es6features
- Today's browsers do not generally support ES6
- The + in ES6+ stands for features from the next version ES7
  - decorators

INTRODUCTION

DEVELOPMENT OPTIONS

- Your code is written in plain JavaScript
- Your code is deployed on the server
- Your code is loaded by the browser



INTRODUCTION

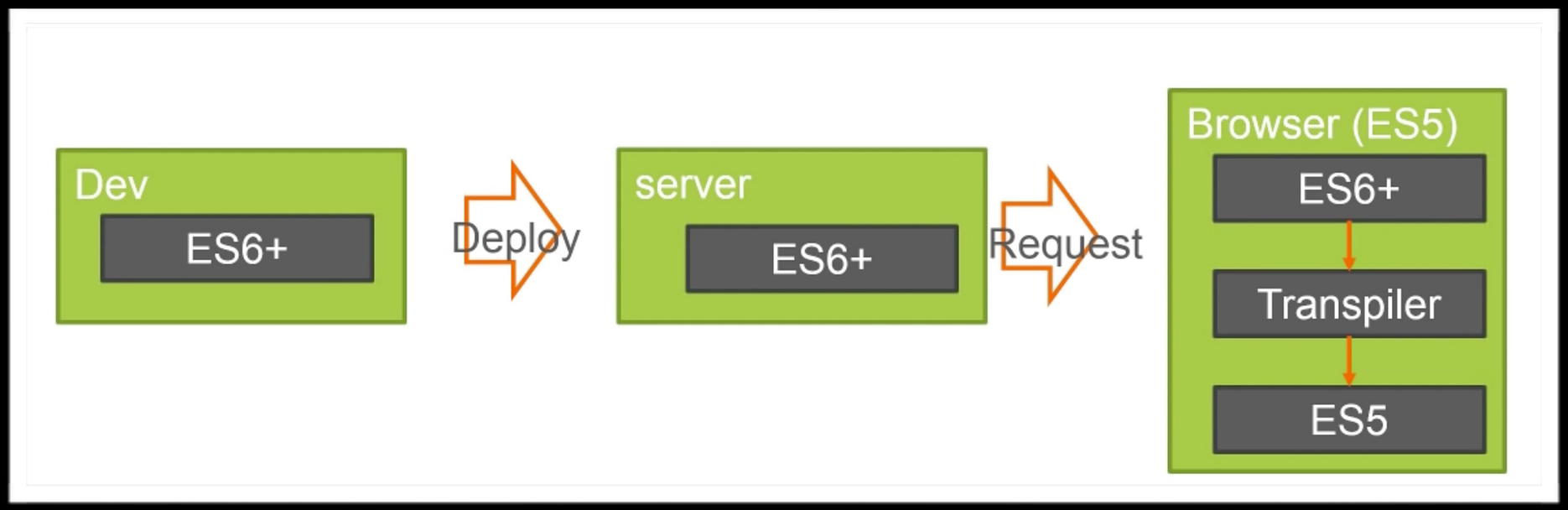
BUILD WITH BUILD-TIME TRANSPIRING

- Your code is written in ES6+
- Your code is transpiled into ES5 and deployed to server
- Your ES5 code is loaded and executed by the browser

INTRODUCTION

BUILD WITH RUN-TIME TRANSPIRING

- Your code is written in ES6+
- Your code is deployed to server
- Your ES6+ code is loaded, transpiled and executed by the browser



DEMO...

HELLO ANGULAR!

CHAPTER

TYPESCRIPT

TYPESCRIPT

TYPES

- Annotate variables, parameters, functions and fields with type information
- Benefits
  - Error during compilation instead of runtime unexpected results
  - Intellisense during development

TYPESSCRIPT

TYPES

- Type annotations in code

```
var height:number = 6;
var isDone:boolean = true;
var name:string = 'thoughtram';

var list:number[] = [1, 2, 3];
var list:Array<number> = [1, 2, 3];

function add(x: number, y: number): number {
  return x+y;
}
```

TYPESCRIPT

CLASSES

- Syntactic sugar for JavaScript prototypes introduced in ES2015.

```
class AppController {

  constructor($http) {
    this.$http = $http;
  }

  doSomething() {}
}

let ctrl = new AppController($http);
```

TYPESCRIPT

CLASSES

- Syntactic sugar for JavaScript prototypes introduced in ES2015.

```
class AppController {

  constructor($http) {
    this.$http = $http;
  }

  doSomething() {}
}

let ctrl = new AppController($http);
```

TYPESCRIPT

CLASSES

- Controllers and Services in Angular are constructor functions

```
class AppController {...}
class BasketService {...}

angular.module('myApp', [])
.controller('AppController', AppController)
.service('BasketService', BasketService);
```

TYPESCRIPT

ARROW FUNCTIONS

- Arrow functions have lexical this and are better suited for defining subroutines.

- This code breaks!

```
app.controller('AppController', function ($http) {
  this.items = [];

  $http.get('items.json').then(function (response) {
    this.items = response.data; // `this` != lexical
  });
});
```

TYPESCRIPT

ARROW FUNCTIONS

- Arrow functions have lexical this and are better suited for defining subroutines.
- This code smells!

```
app.controller('AppController', function ($http) {
  var self = this;
  self.items = [];

  $http.get('items.json').then(function (response) {
    self.items = response.data;
  });
});
```

TYPESCRIPT

ARROW FUNCTIONS

- Arrow functions have lexical this and are better suited for defining subroutines.
- This code rocks!

```
app.controller('AppController', ($http) => {
  this.items = [];

  $http.get('items.json').then((response) => {
    this.items = response.data; // <- lexical this
  });
});
```

TYPESCRIPT

PLAYGROUND

- Handbook
- Interactive Tutorials
- Useful links
  - http://blog.pluralsight.com/extending-classes-and-interfaces-using-typescript
  - http://www.typescriptlang.org/Playground

LABS...

STUDENT TRACK SURVEY TYPESCRIPT

CHAPTER

COMPONENTS

COMPONENTS

WHAT ARE...

- Angular2 is all about components
- Components are
  - the building blocks of the applications
  - self contained
- Components have
  - Logic: Class with methods and state
  - Views: Templates and Styles
- Components tied together form the application

COMPONENTS

WHAT ARE...

```
import {Component} from 'angular2/angular2';

@Component({
  selector: 'contacts-app'
  template: 'Hello World!'
})
class App {

}
```

COMPONENTS

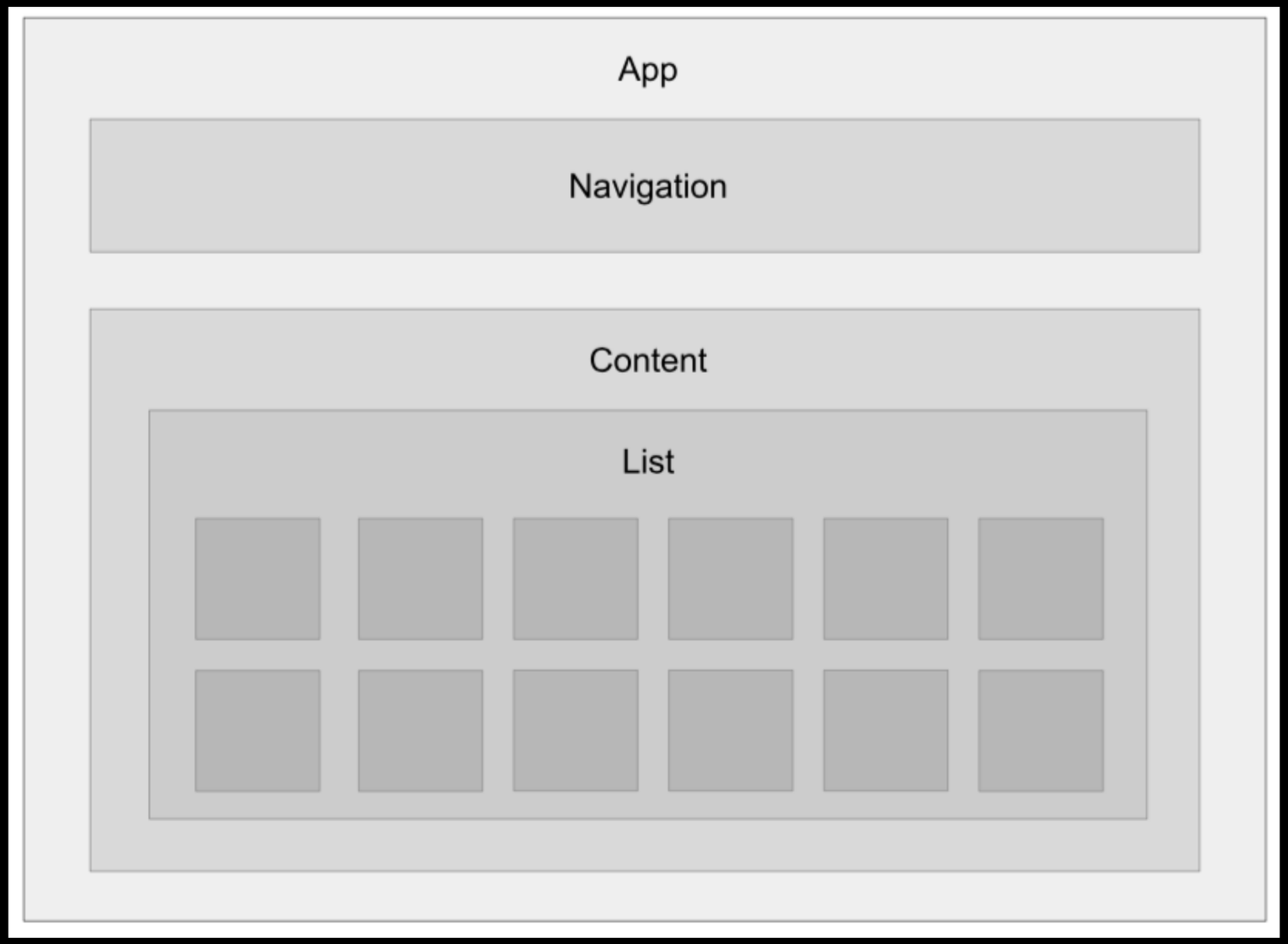
NESTING

```
@Component({
  selector: 'contacts-app',
  directives: [AppHeader, AppNav, AppMain],
  template: '
    <app-header></app-header>
    <app-nav></app-nav>
    <app-main></app-main>
  '
})
class App {

}
```

COMPONENTS

NESTING



COMPONENTS

TEMPLATE SYNTAX

- Template contains HTML -> View of the component
- HTML contains
  - HTML Markup
  - Child components
  - Bindings
  - Template directives
  - Expression operators (Pipes and Elvis) ..more on this later!

COMPONENTS

TEMPLATE SYNTAX

Data Direction	Syntax	Binding Type

One way from data source to view target	`{{expression}}`
	`[target] = "expression"`
	`bind-target = "expression"`
		Interpolation
		Property
		Attribute
		Class
		Style

One way from view target to data source	`(target) = "expression"` `on-target = "expression"`	Event
Two way	`[(target)] = "expression"` `bindon-target = "expression"`	Two way

COMPONENTS

PROPERTIES VS ATTRIBUTES

- HTML attributes are not DOM properties
  - There are attributes that are not properties
  - Attributes are strings
  - There are properties that are not attributed
- Attributes initialise DOM properties
- Angular prefers property bindings!

COMPONENTS

TEMPLATE SYNTAX

Binding Type	Target	Example

Property	Element property
	Component property
	Directive property
		`<img [src] = "heroImageUrl">`
		`<hero-detail [hero]="currentHero">`
		`<div [ng-class] = "{selected: isSelected}">`

Event	Element event
	Component event
	Directive event
		`<button (click) = "onSave()">`
		`<hero-detail (deleted)="onDeleted()">`
		`<input (ng-model-change) = "Name = $event">`

Two way	Directive event Property	`<input [(ng-model)] = "firstName">`
Attribute	Attribute (the exception)	`<button [attr.aria-label] = "actionName">`
Class	Class property	`<div [class.special] = "isSpecial">`
Style	Style property	`<button [style.color] = "isOK ?'red':'green'">`

COMPONENTS

TEMPLATE SYNTAX

- Watch out! properties are lowercased before evaluation

```
<!-- BAD: No Mixed Case -->
<label [textContent] = "title">
<hero-detail [isActive] = “isActive">
```

```
Template parse errors:
Can't bind to 'textcontent' since it isn't a known native
property in ...
```

```
<!-- Lower Snake Case -->
<label [text-content] = "title">
<hero-detail [is-active] = "isActive">
```

According to this rule, we should write [inner-h-t-m-l] to access the element’s innerHTML property. Fortunately, the Angular template parser recognizes inner-html as an acceptable alias for innerHTML.

COMPONENTS

TEMPLATE SYNTAX

- Local variables

```
<!-- phone refers to the input element; pass its `value` to an
   event handler -->

<input #phone placeholder="phone number">
<button (click)="callPhone(phone.value)">Call</button>


<!-- officFax refers to the input element; pass its `value` to
   an event handler -->

<input var-office-fax placeholder="phone number">
<button (click)="callFax(officeFax.value)">Fax</button>
```

Var- and # are alternative notations

COMPONENTS

TEMPLATE DIRECTIVES

- NgIf

```
<div *ng-if="currentHero">Add </div>
<div *ng-if="nullHero">Remove </div>
```

```
<div>Hero Detail removed from DOM because isActive is false</
  div>
```

```
<hero-detail *ng-if="isActive" [hero]="crtHero"></hero-detail>
```

Dont forget to include the CORE_DIRECTIVES in the component

COMPONENTS

TEMPLATE DIRECTIVES

- NgSwitch

```
<div class="toe">You picked

  <span [ng-switch]=“toeChoice(toePicker)">

    <template [ng-switch-when]="'Eenie'">Eenie</template>
    <template [ng-switch-when]="'Meanie'">Meanie</template>
    <template [ng-switch-when]="'Miney'">Miney</template>
    <template [ng-switch-when]="'Moe'">Moe</template>
    <template ng-switch-default>Other</template>

  </span>

</div>
```

Dont forget to include the CORE_DIRECTIVES in the component

COMPONENTS

TEMPLATE DIRECTIVES

- NgFor

```
<div *ng-for="#hero of heroes", #i=index">{{i+1}} -
{{hero.fullName}}</div>
```

We can apply an ng-for repeater to Components as well, as we do in this following example with the LittleHeroComponent:

```
<little-hero *ng-for="#hero of heroes" [hero]="hero"></little-
hero>
```

Dont forget to include the CORE_DIRECTIVES in the component

DEMO...

COMPONENTS

COMPONENTS

INPUTS AND OUTPUTS

- Components can
  - take input like properties
  - generate output in the form of events
  - Bind to these using [] and () template syntax

```
<details [student]="student" (update)="setUpdated()"></details>
```

COMPONENTS

INPUTS AND OUTPUTS

- Remember the bindings to a component:

```
<hero-detail [hero]="crtHero" (deleted)=“onDeleted($event)">
</hero-detail>
```

- The component is build as this:

```
class HeroDetailComponent {

   @Input() hero: Hero;

   @Output() deleted = new EventEmitter();

   onDelete() { this.deleted.next(this.hero); }
}
```

Don’t forget to include the CORE_DIRECTIVES in the component

COMPONENTS

INPUTS AND OUTPUTS

- Another example:

```
@Component({
  selector: 'date-picker',
  inputs: ['date'],
  outputs: ['dateChanged'],
  ...
})
class DatePicker {
  dateChanged = new EventEmitter();
}
```

DEMO...

COMPONENT INPUTS AND OUTPUTS

COMPONENTS

TWO WAY BINDING

- This syntax

```
<input [ng-model]="todo.text"
       (ng-model-change)=“todo.text=$event">
 </input>
```

- Can be shortened using this syntax

```
<input [(ng-model)]="todo.text"></input>
```

COMPONENTS

TWO WAY BINDING

- Because of this directive

```
@Directive({
  selector: '[ng-model]',

  host: {
    "[value]": 'ngModel',
    "(input)": "ngModelChange.next($event.target.value)"
  }

})
class NgModelDirective {

  @Input() ngModel:any; // stored value
  @Output() ngModelChange:EventEmitter; = new EventEmitter()
}
```

- Exercise: explain this directive to me!

LABS...

STUDENT TRACK SURVEY COMPONENTS

CHAPTER

PIPES

PIPES

USING PIPES

- Filters in AngularJS, format and transform output
- Examples

```
<p>The hero's birthday is {{ birthday | date }} </p>
```

```
<p>The hero's birthday is {{ birthday | date:"MM/dd/yy" }} </p>
```

```
<p>The hero's birthday {{ birthday | date | uppercase}} </p>
```

```
<p>
The hero's birthday {{ ( birthday | date:'fullDate' ) | uppercase}}
</p>
```

PIPES

BUILTIN PIPES

Filter	Angularjs 1.X	Angular2
Number	X	-
Orderby	X	-
Filter	X	-
Currency	X	X
Date	X	X
Uppercase	X	X
Lowercase	X	X
Json	X	X
Limitto	X	X
Async	-	X
Decimal	-	X
Percent	-	X
Slice	-	X

DEMO...

BUILTIN PIPES

PIPES

CUSTOM PIPES

- @Pipe annotation
- Transform method
- Angular reports an error if we neglect to list our custom pipe. All Angular built-in pipes are pre-registered.
- Custom pipes must be registered manually!

PIPES

CUSTOM PIPES

- Simple pipe definition

```
import {bootstrap, Component, Pipe} from 'angular2/angular2'

@Pipe({
  name: 'exponentialStrength'
})
class ExponentialStrengthPipe {

  transform(value:number, args:string[]) : any {
    return Math.pow(value, parseInt(args[0] || 1, 10));
  }
}
```

PIPES

CUSTOM PIPES

- Simple pipe usage

```
@Component({
  selector: 'power-booster',
  template: `
    <p>
      Super power boost: {{2 | exponentialStrength: 10}}
    </p>
  `,
  pipes: [ExponentialStrengthPipe]
})
class PowerBooster { }

bootstrap(PowerBooster);
```

PIPES

STATEFULL PIPES

- Pipes who's output is checked each cycle, even when input did not change
- Async pipe is statefull

PIPES

STATEFULL PIPES

- Async pipe: bind our templates directly to values that arrive asynchronously.

```
@Component({
  selector: 'pipes',
  template: '<h1>Async</h1><p>{{ promise | async}}</p>',
})
class PipesAppComponent {
  promise: Promise;
  constructor() {
    this.promise = new Promise(function(resolve, reject) {
      setTimeout(function() {
        resolve("Hey, I'm from a promise.");
      }, 2000)
    });
  }
}
```

- great for working with promises and observables

PIPES

STATEFULL PIPES

- custom stateful pipe

```
@Pipe({
  name: 'fetch',
  pure: false
})
class FetchJsonPipe {
  private fetchedValue:any;
  private fetchPromise:Promise<any>;
  transform(value:string, args:string[]):any {
    if (!this.fetchPromise) {
      this.fetchPromise = fetch(value)
        .then(result => result.json())
        .then(json => { this.fetchedValue = json; });
    }
    return this.fetchedValue;
  }
}
```

DEMO...

CUSTOM PIPES

LABS...

STUDENT TRACK SURVEY PIPES

CHAPTER

SERVICES

SERVICES

INTRODUCTION

- Dependency injection
- Http service
- Custom services

SERVICES

DEPENDENCY INJECTION

- Application wide
- Singleton
- Typescript emits metadata for decorated classes
- Angular DI uses metadata for resolving
- implicit or explicit injected

```
import {Hero} from './hero';
import {HEROES} from './mock-heroes';
class HeroService {
  heroes: Hero[];
  constructor() {
    this.heroes = HEROES;
  }
  getHeroes() { return this.heroes; }
}

bootstrap(AppComponent, [HeroService]);
```

```
 constructor(heroService: HeroService) {
   this.heroes = heroService.getHeroes();
 }
// OR //
 constructor(@Inject(HeroService)
                          heroService) {
   this.heroes = heroService.getHeroes();
 }
```

SERVICES

DEPENDENCY INJECTION

- Per component
- Providers array in decorator
- implicit or explicit injected
- More on http later..

```
import {Http, HTTP_PROVIDERS} from '..';
@Component({
  selector: 'http-app',
  providers: [HTTP_PROVIDERS],
  templateUrl: 'people.html'
})

bootstrap(AppComponent);

 constructor(http: Http) {
   …
 }
// OR //
 constructor(@Inject(Http) http) {
   …
 }
```

SERVICES

DEPENDENCY INJECTION

- DI makes testing components easy

```
it("should have heroes when created", () => {
  let hc = new HeroesComponent(mockService);
  expect(hc.heroes.length).toEqual(mockService.getHeroes().length);
})
```

SERVICES

DEPENDENCY INJECTION

- What if a service needs a service??
- Decorator to hold metadata, DI needs metadata

```
import {Hero} from './hero';
import {HEROES} from './mock-heroes';
import {Logger} from ‘./logger';

@Injectable()
class HeroService {
  heroes: Hero[];
 constructor(private logger: Logger) {
    this.heroes = HEROES;
  }
  getHeroes() {
    this.logger.log('Getting heroes ...')
    return this.heroes;
  }
}
```

SERVICES

INJECTOR PROVIDERS

- What happens in bootstrap?

```
bootstrap(AppComponent, [HeroService]);
```

```
is shorthand for
```

```
bootstrap(AppComponent, [ provide(HeroService, {useClass:HeroService})];
```

```
which evaluates to
```

```
bootstrap(AppComponent, [new Provider(HeroService,
{useClass:HeroService})]);
```

SERVICES

INJECTOR PROVIDERS

- So we can use this with a custom service class…

```
bootstrap(AppComponent,
  [new Provider(HeroService, {useClass:EmptyHeroService})]);
```

- But also with a factory class…

```
let hSF = (logger: log, userService: usr) => {
  return new HeroService(logger, userService.user.isSpecial);
}
```

```
bootstrap(AppComponent,
   [ new Provider(HeroService,
          {useFactory:hSF, deps:[log,usr]}), log, usr]);
```

SERVICES

INJECTOR PROVIDERS

- Injecting configuration data

```
let config = {
  apiEndpoint: 'api.heroes.com',
  title: 'The Hero Employment Agency'
};
```

```
bootstrap(AppComponent, [ //others//,
    provide('App.config', {useValue:config})
]);
```

```
import {Inject} from 'angular2/angular2'
```

```
constructor(heroService: HeroService, @Inject('app.config') config)
```

SERVICES

HTTP SERVICE

- http service uses observables!

```
import {Http, HTTP_PROVIDERS} from 'angular2/http';
@Component({
  selector: 'http-app',
  providers: [HTTP_PROVIDERS],
  templateUrl: 'people.html'
 })

class PeopleComponent {
     people:any;
     constructor(http: Http) {
     http.get('people.json')
        .map(res => res.json())
        .subscribe(people => this.people = people);
   }
 }
```

SERVICES

CUSTOM SERVICE

- Is just a class

```
import {Hero} from './hero';
import {HEROES} from './mock-heroes';

class HeroService {
  heroes: Hero[];
  constructor() {
    this.heroes = HEROES;
  }
  getHeroes() {
    return this.heroes;
  }
}

bootstrap(AppComponent, [HeroService]);
```

DEMO...

SERVICES

LABS...

STUDENT TRACK SURVEY SERVICES

CHAPTER

FORMS

FORMS

FEATURES

- Two-way data binding with ***ng-model**
- Track change-state and validity with **ng-control** and apply CSS classes.
  - control visited: (ng-touched | ng-untouched)
  - value changed: (ng-pristine | ng-dirty)
  - validity: (ng-valid | ng-invalid)
- validation
- error handling

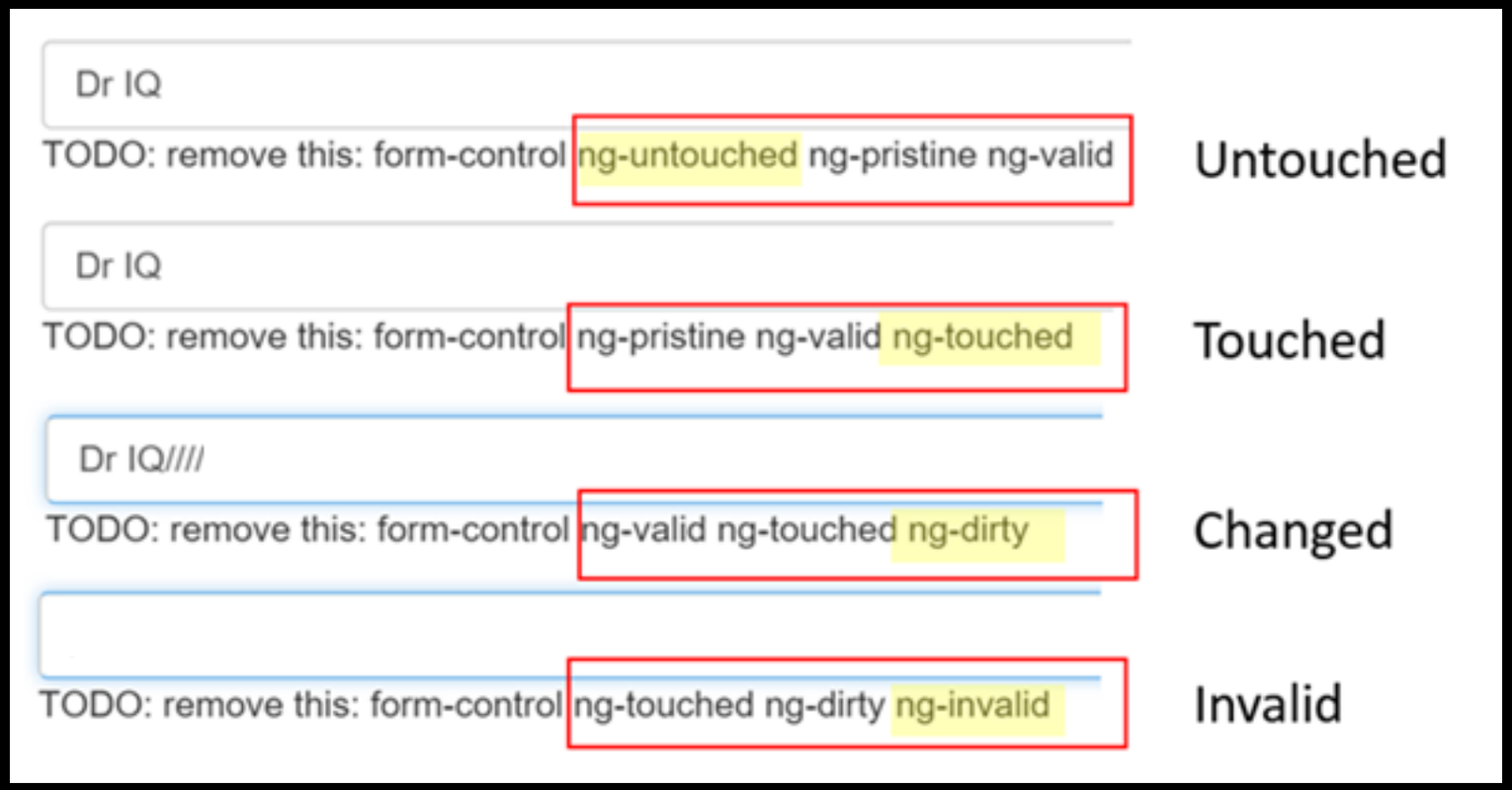
FORMS

FEATURES

- Track change-state with **ng-control**

```
<input #spy type="text" required
  [(ng-model)]="model.name"
    ng-control="name"  >

TODO: remove this: {{spy.className}}
```



FORMS

FEATURES

- Show validation error messages

```
<input type="text" class="form-control" required
  [(ng-model)]="model.name"
    ng-control="name"  #name="form" >
<div [hidden]="name.valid" class="alert alert-danger">
  Name is required
</div>
```

FORMS

EXAMPLE

```
import {Component, bootstrap, View} from "angular2/angular2";
import {FORM_DIRECTIVES, FormBuilder, ControlGroup} from "angular2/angular2";

@Component({
  selector: 'demo-form-sku'
  directives: [FORM_DIRECTIVES],
  template: `<div><h2>Demo Form: Sku</h2>
    <form #f=“form" (submit)="onSubmit(f.value)">
      <div class="form-group">
        <label for="skuInput">SKU</label>
        <input type="text" class="form-control"
               id=“skuInput" placeholder=“SKU" ng-control="sku">
      </div>
      <button type="submit" class="btn btn-default">Submit</button>
  </form></div>`
})
export class DemoFormSku {
  onSubmit(value) { console.log('you submitted value: ', value);}
}
```

FORMS

TEMPLATE DRIVEN FORMS

- The template

```
<form #f="form" (ng-submit)="onSubmitTemplateBased()">
    <p>
        <label>First Name:</label>
        <input type="text" ng-control="firstName"
            [(ng-model)]="vm.firstName" required>
    </p>
    <p>
        <label>Password:</label>
        <input type="password" ng-control="password"
            [(ng-model)]="vm.password" required>
    </p>
    <p><button type="submit" [disabled]="!f.valid">Submit</button></p>
</form>
```

FORMS

TEMPLATE DRIVEN FORMS

- The component

```
@Component({
    selector: "template-driven-form"
})
@View({
    templateUrl: 'template-driven-form.html',
    directives: [FORM_DIRECTIVES]
})
export class TemplateDrivenForm {
    vm: Object = {};
    onSubmitTemplateBased() {
        console.log(this.vm);
    }
}
```

FORMS

MODEL DRIVEN FORMS

- The template

```
<form [ng-form-model]="form" (ng-submit)="onSubmit()">
    <p>
        <label>First Name:</label>
        <input type="text" ng-control="firstName">
    </p>
    <p>
        <label>Password:</label>
        <input type="password" ng-control="password">
    </p>
    <p>
        <button type="submit" [disabled]="!form.valid">Submit</button>
    </p>
</form>
```

FORMS

MODEL DRIVEN FORMS

- The component

```
@Component({
    selector: "model-driven-form"
    templateUrl: 'model-driven-form.html',
    directives: [FORM_DIRECTIVES]
})
export class ModelDrivenForm {
    form: ControlGroup;
    firstName: Control = new Control("", Validators.required);
    constructor(fb: FormBuilder) {
        this.form = fb.group({
            "firstName": this.firstName,
            "password":["", Validators.required]
        });
    }
    onSubmitModelBased() {
       console.log(this.form);
    }
}
```

DEMO...

FORMS

LABS...

STUDENT TRACK SURVEY FORMS

CHAPTER

ROUTING

ROUTING

STEPS

- Include script for routing

```
<html>
<head>
    <script src="../node_modules/systemjs/dist/system.src.js"></script>
    <script src="../node_modules/angular2/bundles/angular2.dev.js"></script>
    <script src="../node_modules/angular2/bundles/router.dev.js"></script>

    <link rel="stylesheet" href="./palette.css"></link>
    <base href="/src/" />
</head>
<body>
    <studenttrack-survey></studenttrack-survey>
</body>
```

ROUTING

STEPS

- Change the application component
- (Full page example)

```
1  import {Component, bootstrap, CORE_DIRECTIVES, FORM_DIRECTIVES} from 'angular2/angular2';
2  import {RouteConfig, RouterLink, RouterOutlet, ROUTER_PROVIDERS} from 'angular2/router';
3  import {StudentDetails} from './components/studentdetails';
4  import {student, studenttrack} from './models/student';
5  import { StudentService } from './services/studentservice';
6  import { StudentTrackService } from './services/studenttrackservice';
7  import { StudentCard } from './components/studentcard';
8  import { StudentForm } from './form';
9
10 @Component({
11     selector: 'studenttrack-survey',
12     directives:[FORM_DIRECTIVES, CORE_DIRECTIVES, StudentDetails, StudentForm, RouterLink, RouterOutlet],
13     providers:[StudentService, StudentTrackService, ROUTER_PROVIDERS],
14     template: `
15     <student-form></student-form>
16     <div *ng-for="#studenttrack of studenttracks.getStudentTracks()" class="studenttrack light-primary-color text
17        <h1 class="dark-primary-color text-primary-color">Studenttrack {{studenttrack.name}} (<span [text-content]
18             <studentdetails *ng-for="#student of studenttrack.getStudents()"
19                  [router-link]="['/Student', {id: student.id }]" [student]="student" [isSelected]="currentstudent
20             </studentdetails>
21      </div>
22      <div style="margin:5px;"><router-outlet></router-outlet> </div>
23     `,
24     styles:[`
25      .studenttrack { border:1px solid black;margin:5px;padding:0px; }
26      .studenttrack h1 { margin:0px;padding:15px;}
27     `]
28 })
29 @RouteConfig([
30     { path:'/', redirectTo:'/home'},
31     { path:'/src/index/:id', as:'Student', component:StudentCard}
32 ])
33 class SurveyApplication {
34     studenttracks:StudentTrackService;
```

```
// YOUR IMPORTS HERE...
import {Component, EventEmitter, NgClass} from 'angular2/angular2';
import { RouterLink, RouteParams } from 'angular2/router';
import {student} from '../models/student';
import { StudentFormatter } from '../pipes/studentFormatter';
import { StudentService } from '../services/studentservice';

@Component({
    selector:'student-card',
    inputs:['student'],
    directives:[NgClass],
    pipes:[StudentFormatter],
    template:`
    <div class="dialog light-primary-color text-primary-color"><div [ng-class]="{defaultPrimaryColor
    {{ student.firstname }} {{ student.lastname }}</div>
    <p> Address: {{ student.address }} <br/> Phone: {{ student.phone }} </p></div>
    `,
    styles:[`
    .dialog { border:1px solid black;};
    .header { margin-top:0px;}
    `]
})
export class StudentCard
{
    student:student;
    constructor(studentservice:StudentService, routeParams:RouteParams){
        this.student = studentservice.getStudentById(routeParams.params.id);

    }
}
```

DEMO...

ROUTING

THATS ALL!

THANK YOU!

JOHN.GORTER@INFOSUPPORT.COM
TWITTER: @JOHNGORTER