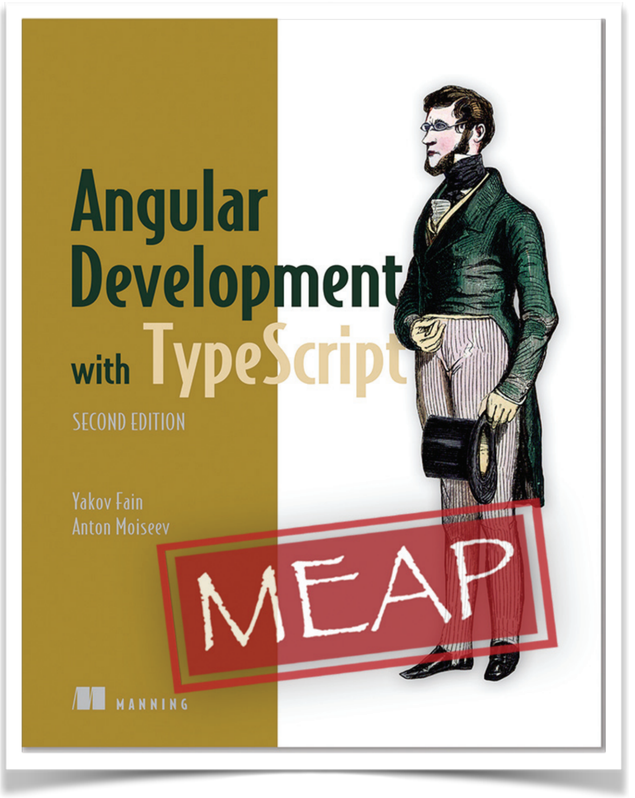
Angular Applied

Yakov Fain

Farata Systems

About myself

- Work for Farata Systems
  Angular consulting
- Java Champion
- Working on the 2nd edition of:
  “Angular Development with TypeScript”



For 42% off
use code slfain
at http://bit.ly/2ybDA7f

The Agenda

- Jump-start a project with Angular CLI
- Major concepts:

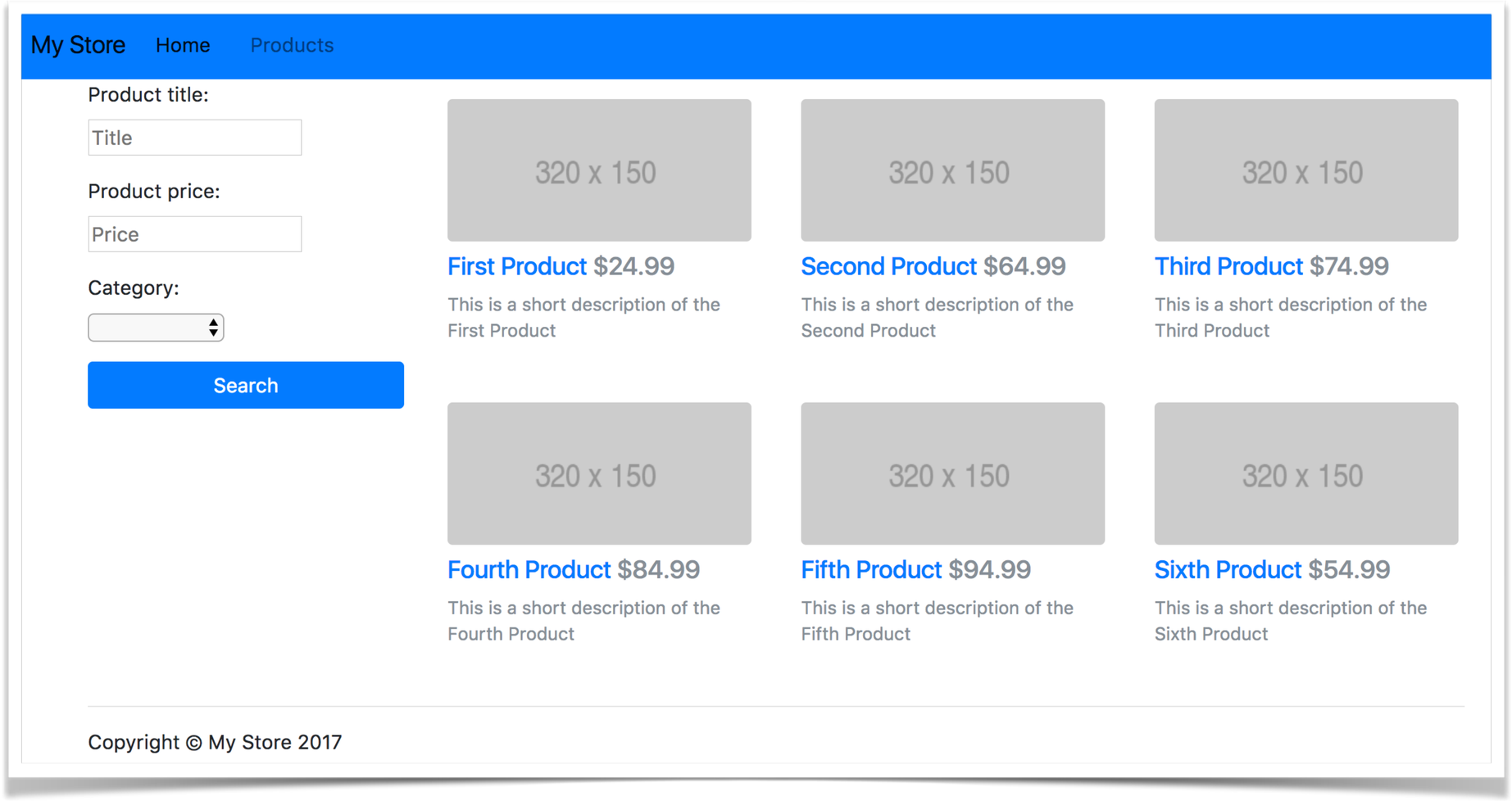
  Components, services, DI, binding,

  routing, inter-component communications
- Apply the concepts by creating a myStore app
- Build and deploy the app on a web server

Angular

- Component-based (not an MVC framework)
- Declarative templates
- Dependency Injection
- Router
- Integrated RxJS
- Can write apps in TypeScript or JavaScript (ES5 or ES6)

The landing page of myStore



Main artifacts

- **Component** - a class that includes UI (a template)
- **Directive** - a class with the code to perform actions on components but has no UI of its own
- **Service** - a class where you write business logic
- **Pipe** - a transformer function that can be used in templates
- **Module** - a class that lists all of the above

Angular module

Other modules if needed

```
import { BrowserModule } from ‘@angular/platform-browser';

import { NgModule } from '@angular/core';

import { FormsModule } from '@angular/forms';
import { HttpClientModule } from ‘@angular/common/http';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],

  bootstrap: [AppComponent]
})
export class AppModule { }
```

All module components

Root component

Installing Angular and dependencies

npm: Node Package Manager

- Install NodeJS form https://nodejs.org
- npm comes with Node.js
- The npmjs.org repository has 400K+ packages
- All packages required by Angular apps are installed from npmjs.org

Semantic Versioning

https://docs.npmjs.com/misc/semver

5.0.1

Major
breaking
changes

Minor
new features,
not breaking

Patch
bug fixes,
not breaking

package.json

Required for dev and prod

Required for dev

```
"dependencies": {
  "@angular/animations": "^5.0.0",
  "@angular/common": "^5.0.0",
  "@angular/compiler": "^5.0.0",
  "@angular/core": "^5.0.0",
  "@angular/forms": "^5.0.0",
  "@angular/http": "^5.0.0",
  "@angular/platform-browser": "^5.0.0",
  "@angular/platform-browser-dynamic": "^5.0.0",
  "@angular/router": "^5.0.0",
  "core-js": "^2.4.1",
  "rxjs": "^5.5.2",
  "zone.js": "^0.8.14"
},
"devDependencies": {
  "@angular/cli": "1.5.0",
  "@angular/compiler-cli": "^5.0.0",
  "@angular/language-service": "^5.0.0",
  "@types/jasmine": "~2.5.53",
  "@types/jasminewd2": "~2.0.2",
  "@types/node": "~6.0.60",
  "codelyzer": "~3.2.0",
  "jasmine-core": "~2.6.2",
  "jasmine-spec-reporter": "~4.1.0",
  "karma": "~1.7.0",
  "karma-chrome-launcher": "~2.1.1",
  "karma-cli": "~1.0.1",
  "karma-coverage-istanbul-reporter": "^1.2.1",
  "karma-jasmine": "~1.1.0",
  "karma-jasmine-html-reporter": "^0.2.2",
  "protractor": "~5.1.2",
  "ts-node": "~3.2.0",
  "tslint": "~5.7.0",
  "typescript": "~2.4.2"
}
```

Angular CLI features

- Scaffolding the project and creating a basic app
- Generating components, services, modules, etc.
- Dev web server
- Supports dev and prod builds
- Configuring test runners

Getting started with Angular CLI

- Install Angular CLI globally

  npm i @angular/cli -g

- Generate new project using the command **ng new**

Running a newly generated project

- Generate a new Angular project with **ng new**:

```
ng new myStore
```

- Change to your project dir:

```
cd myStore
```

- Build the app bundles in memory and open the app in the browser on port 4200:

```
ng serve -o
```

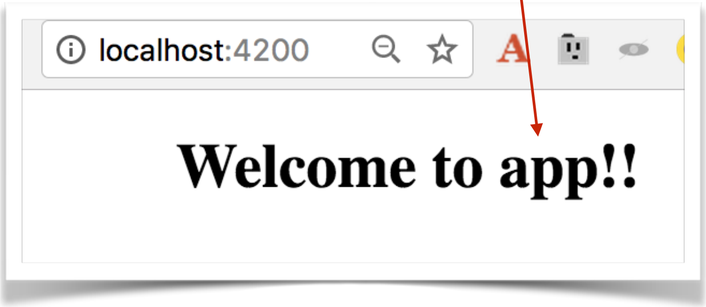
Generated app.component.ts: and app.component.html

app.component.ts

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

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

app.component.html

```html
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!!
  </h1>
  </div>
...
```



ng generate (ng g)

- Component: **ng g c**
- Service: **ng g s**
- Module: **ng g m**
- Directive: **ng g d**
- Help: **ng help generate**

UI libraries and layout options

1. Bootstrap and ng-bootstrap
2. Angular Material
3. Angular Flex Layout
4. PrimeNG
5. ... and more

Hands-on

Generating myStore and its artifacts

(steps 1.1 - 1.3)

Instructions in myStore_3hours.html
at https://github.com/yfain/mystore

TypeScript in 10 min

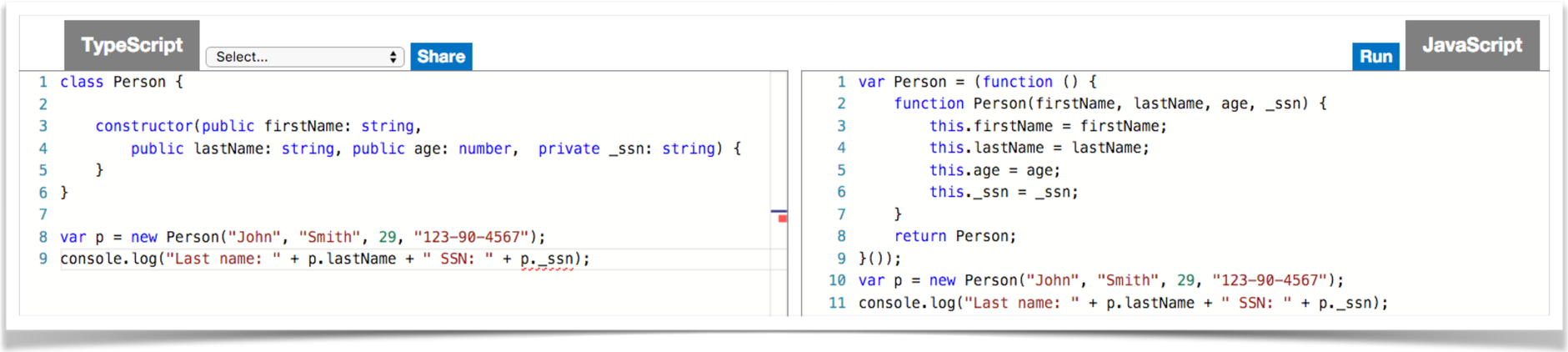
Watch my video course “TypeScript Essentials”
on safaribooksonline.com http://bit.ly/2fp5OF5

What's TypeScript

- A superset of JavaScript
- Increases your productivity in writing JavaScript
- Supports types, classes, interfaces, generics, annotations
- Supports most of the ES6 - ES8 syntax

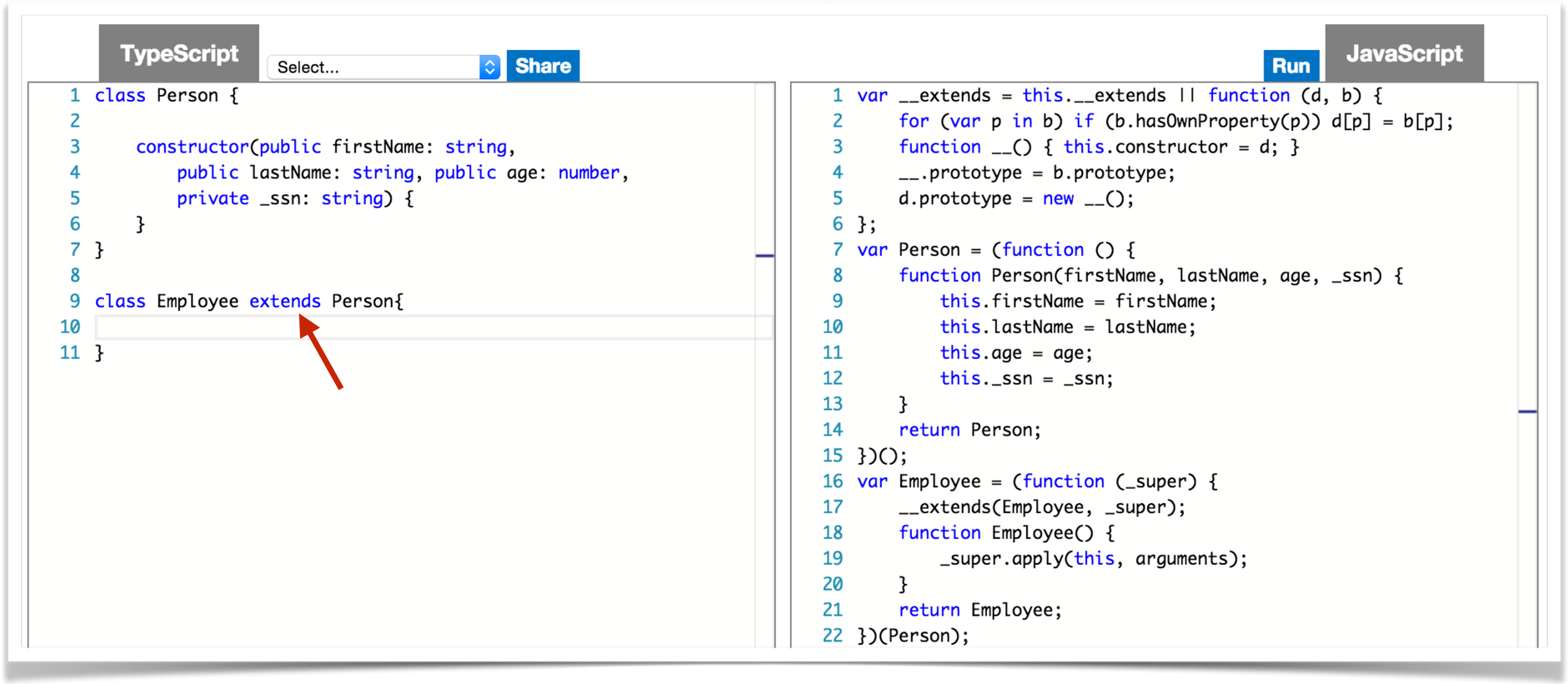
A Class With Constructor

TypeScript → JavaScript (ES5)



Inheritance

Classical syntax → Prototypal



TypeScript | Select... | Share

```
1  class Person {
2
3      constructor(public firstName: string,
4          public lastName: string, public age: number,
5          private _ssn: string) {
6      }
7  }
8
9  class Employee extends Person{
10
11 }
```

Run | JavaScript

```
1  var __extends = this.__extends || function (d, b) {
2      for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
3      function __() { this.constructor = d; }
4      __.prototype = b.prototype;
5      d.prototype = new __();
6  };
7  var Person = (function () {
8      function Person(firstName, lastName, age, _ssn) {
9          this.firstName = firstName;
10         this.lastName = lastName;
11         this.age = age;
12         this._ssn = _ssn;
13     }
14     return Person;
15 })();
16 var Employee = (function (_super) {
17     __extends(Employee, _super);
18     function Employee() {
19         _super.apply(this, arguments);
20     }
21     return Employee;
22 })(Person);
```

Generics

TypeScript | Select... | Share

```typescript
class Person {
    name: string;
}

class Employee extends Person{
    department: number;
}

class Animal {
    breed: string;
}

var workers: Array<Person> = [];

workers[0] = new Person();
workers[1] = new Employee();
workers[2] = new Animal();
```

Compile time error

Run | JavaScript

```javascript
var __extends = (this && this.__extends) || function (d, b) {
    for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
    function __() { this.constructor = d; }
    d.prototype = b === null ? Object.create(b) : (__.prototy
};
var Person = (function () {
    function Person() {
    }
    return Person;
})();
var Employee = (function (_super) {
    __extends(Employee, _super);
    function Employee() {
        _super.apply(this, arguments);
    }
    return Employee;
})(Person);
var Animal = (function () {
    function Animal() {
    }
    return Animal;
})();
var workers = [];
workers[0] = new Person();
workers[1] = new Employee();
workers[2] = new Animal();
```

No Errors

Interfaces as Custom Types

TypeScript | Select... | Share

```typescript
interface IPerson {

    firstName: string;
    lastName: string;
    age: number;
    ssn?: string;
}

class Person  {

    constructor(public config: IPerson) {
    }
}

var aPerson: IPerson = {
    firstName: "John",
    lastName: "Smith",
    age: 29
}

var p = new Person(aPerson);
console.log("Last name: " + p.config.lastName );
```

Run | JavaScript

```javascript
var Person = (function () {
    function Person(config) {
        this.config = config;
    }
    return Person;
})();
var aPerson = {
    firstName: "John",
    lastName: "Smith",
    age: 29
};
var p = new Person(aPerson);
console.log("Last name: " + p.config.lastName);

```

No interfaces here

Interfaces and `implements`

```typescript
interface IPayable{

  increasePay(percent: number): void
}

class Employee implements IPayable{

    increasePay(percent: number): void {
        // increase salary
    }
}

class Contractor implements IPayable{

    increasePay(percent: number): void {
        // increase hourly rate
    }
}

var workers: Array<IPayable> = [];
workers[0] = new Employee();
workers[1] = new Contractor();

workers.forEach(worker => worker.increasePay(30));
```

```javascript
var Employee = (function () {
    function Employee() {
    }
    Employee.prototype.increasePay = function (percent) {
        // increase salary
    };
    return Employee;
})();
var Contractor = (function () {
    function Contractor() {
    }
    Contractor.prototype.increasePay = function (percent) {
        // increase hourly rate
    };
    return Contractor;
})();
var workers = [];
workers[0] = new Employee();
workers[1] = new Contractor();
workers.forEach(function (worker) { return worker.increasePay(30); });

```

No interfaces
in JavaScript

TypeScript Compiler: tsc

- Install the typescript compiler **tsc** globally:

  npm install typescript -g

- Compile **main.ts** into **main.js** specifying ES5 as target syntax:

  tsc --t ES5 main.ts

Sample tsconfig.json

```
{
  "compilerOptions": {
    "outDir": "./dist",
    "moduleResolution": "node",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "es5",
  }
}
```

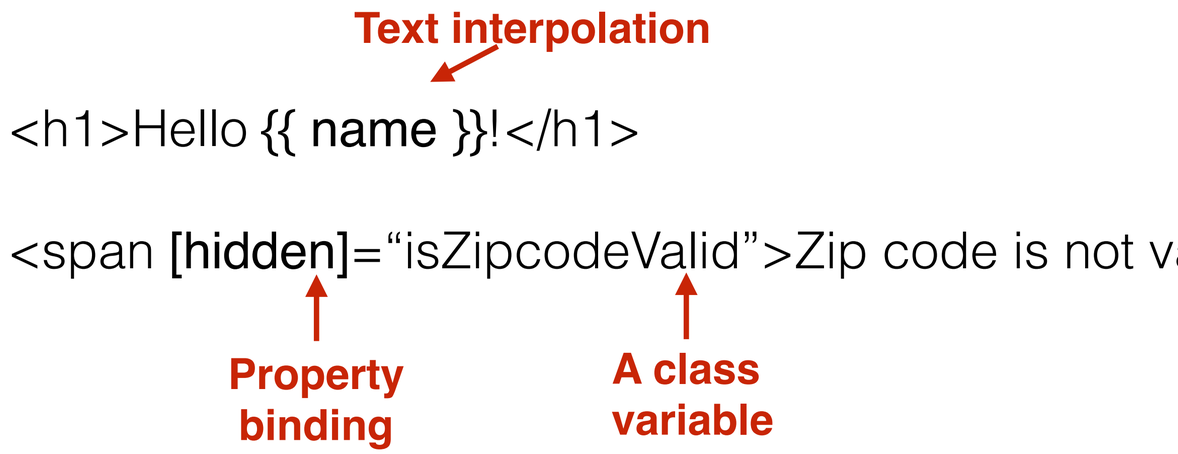
Run **tsc** from dir where tsconfig.json is located

Data Binding

- Keeping the view and data in sync
- UI elements can be bound to component properties
- Events can be bound to functions or expressions

Unidirectional Binding

From code to template:



Unidirectional Binding

From template to code:

```
<button (click)="placeBid()">Place Bid</button>
```


```
<input (input)=“onInputEvent()” />
```


```
<price-quoter (lastPrice)="priceQuoteHandler($event)">
</price-quoter>
```

Dependency Injection

Dependency Injection in Angular

- Angular injects values into components via constructors
- Each component has **its own injector**
- You specify **a provider** so Angular knows **what** to inject
- If a provider specified in @NgModule, you'll get a singleton object

How to declare providers

Providers allows you to map a token to a concrete implementation.

- Defining a provider on the app level

```
@NgModule({
 …
 providers:[{provide:ProductService, useClass:ProductService}]
})
```

- Or when the token and implementation names are the same:

```
@NgModule({
 …
 providers:[ProductService]
})
```

Injection and Providers

- When the token and a type have the same name, use the shorter notation:

```
@Component({
  …
  providers:[ProductService]
}}
```

- Angular injects values via the component's constructor:

```
constructor(productService: ProductService){
      productService.doSomething();
}
```

- If a component doesn't declare a provider for an injectable value, Angular checks component's parent, then grandparent, etc.

Demo: generate a service with Angular CLI

- In the Terminal window run this:

  ng g s product -m app.module

- product.service.ts is generated and its provider is added to app.module.ts

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

@Injectable()
export class ProductService {

  constructor() { }

}
```

```
...
import { ProductService } from './product.service';

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

A sample injectable service

```
export class ProductService {
  getProducts(): Product {
    // An HTTP request can go here
    return new Product( 0, "iPhone 7", 249.99, "The latest iPhone, 7-inch screen");
  }
}
```

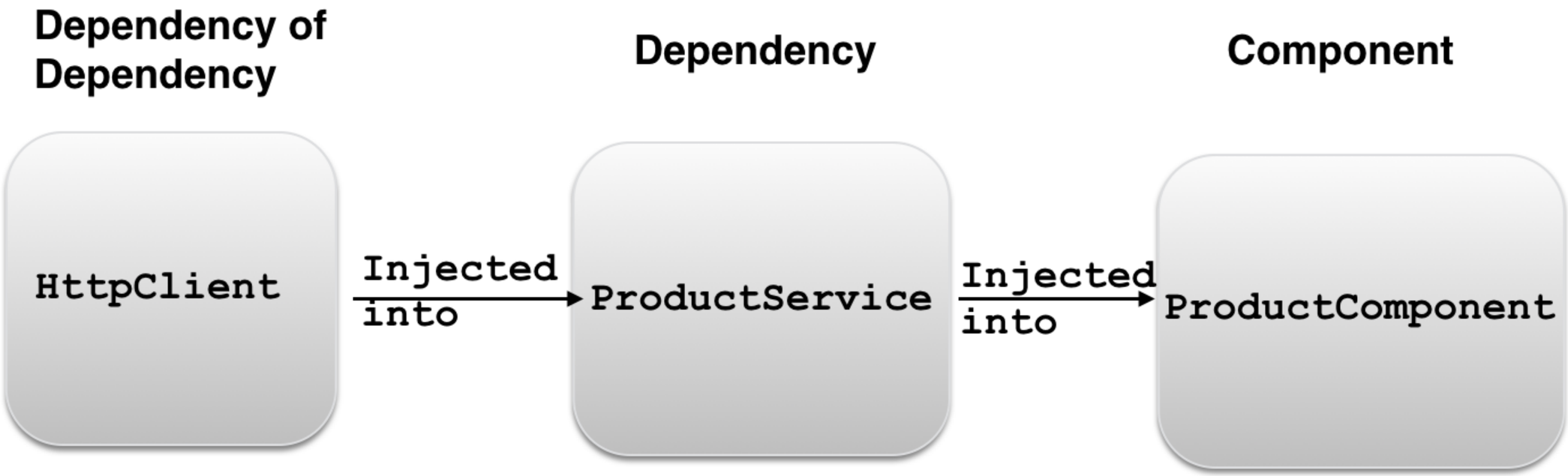
Value Object 

```
export interface Product {

    public id: number;
    public title: string;
    public price: number;
    public description: string;

}
```

Injecting ProductService

```
import {Component} from '@angular/core';
import {ProductService, Product} from "./product.service";

@Component({
  selector: 'di-product-page',
  template: `<div>
  <h1>Product Details</h1>
  <h2>Title: {{product.title}}</h2>
  <h2>Description: {{product.description}}</h2>
  <h2>Price: \${{product.price}}</h2>
</div>`,
  providers:[ProductService]
})

export class ProductComponent {
  product: Product;

  constructor( productService: ProductService) {
   this.product = productService.getProduct();
  }
}
```


Dependencies of dependencies



Dependencies of dependencies

Http providers

```
import {HttpClientModule} from '@angular/common/http';
…


@NgModule({
    imports:      [ BrowserModule, HttpModule],
    declarations: [ AppComponent,
                    ProductComponent],
    bootstrap:    [ AppComponent ]
})
class AppModule { }
```

```
export default class ProductComponent {

  product: Product;

  constructor( productService: ProductService) {

 }
}
```

DI

```
import {HttpClient} from '@angular/common/http';

@Injectable()
export class ProductService {

   constructor(private http:HttpClient){

    let products = this.http.get('products.json');
   }
  // other app code goes here
}
```

DI

Hands-on

Implementing myStore landing page

(steps 1.4 - 1.7)

Inter-component communication

Input and Output Properties

- Think of a component as a black box with entry and exit doors
- Properties marked as **`@Input()`** are used for getting data from the parent component
- The parent component can pass data to its child using bindings to input properties
- Properties marked as **`@Output()`** are used for sending events (and data) from a component

Parent binds to input props

```
@Component({
    selector: 'app',
    template: `
    <input type="text" placeholder="Enter stock (e.g. AAPL)"
                       (change)="onInputEvent($event)">
    <br/>


    <order-processor [stockSymbol]="stock" [quantity]="100">
    </order-processor>
  `
})
class AppComponent {
    stock:string;

    onInputEvent({target}):void{
        this.stock=target.value;
    }
}
```

Input Properties in Child

```
@Component({
    selector: 'order-processor',
    template: `
    Buying {{quantity}} shares of {{stockSymbol}}
    `})
class OrderComponent {

    @Input() quantity: number;

    private _stockSymbol: string;

    @Input()
    set stockSymbol(value: string) {

    if (value != undefined) {
      this._stockSymbol = value;
      console.log(`Sending a Buy order to NASDAQ: ${this.stockSymbol}  ${this.quantity}`);
    }
}
    get stockSymbol(): string {
        return this._stockSymbol;
    }
}
```

Output properties in a child

```
@Component({
    selector: 'price-quoter',
    template: `<strong>Inside PriceQuoterComponent:
                        {{stockSymbol}} {{price | currency:'USD':true}}</strong>`,

    styles:[`:host {background: pink;}`]
})
class PriceQuoterComponent {

    @Output() lastPrice: EventEmitter <IPriceQuote> = new EventEmitter();

    stockSymbol: string = "IBM";
    price:number;

    constructor() {
        setInterval(() => {
            let priceQuote: IPriceQuote = {
                stockSymbol: this.stockSymbol,

                latestPrice: 100*Math.random()
            };

            this.price = priceQuote.latestPrice;

            this.lastPrice.emit(priceQuote);

        }, 1000);
    }
}
```

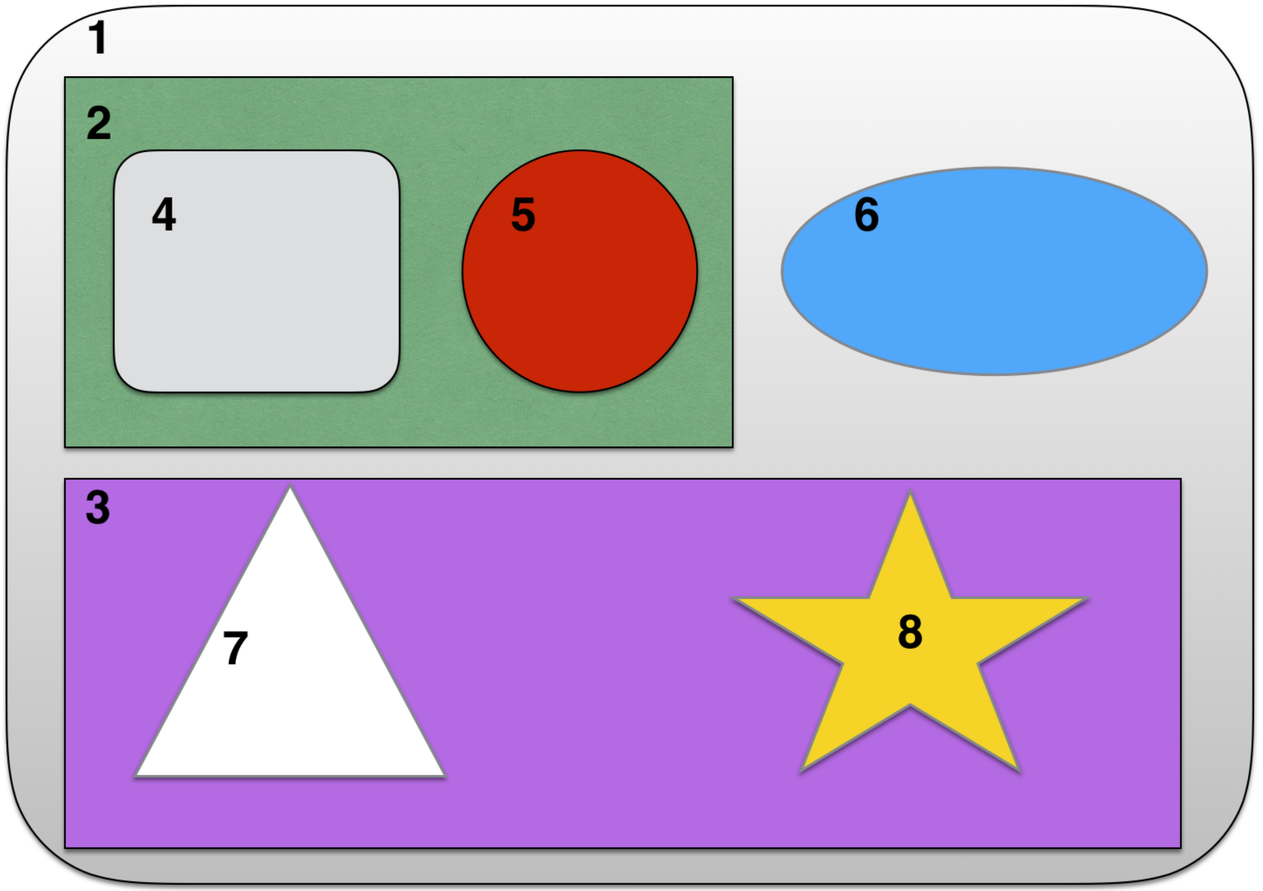
pipe

```
interface IPriceQuote {
    stockSymbol: string;
    latestPrice: number;
}
```

The parent listens to the lastPrice event

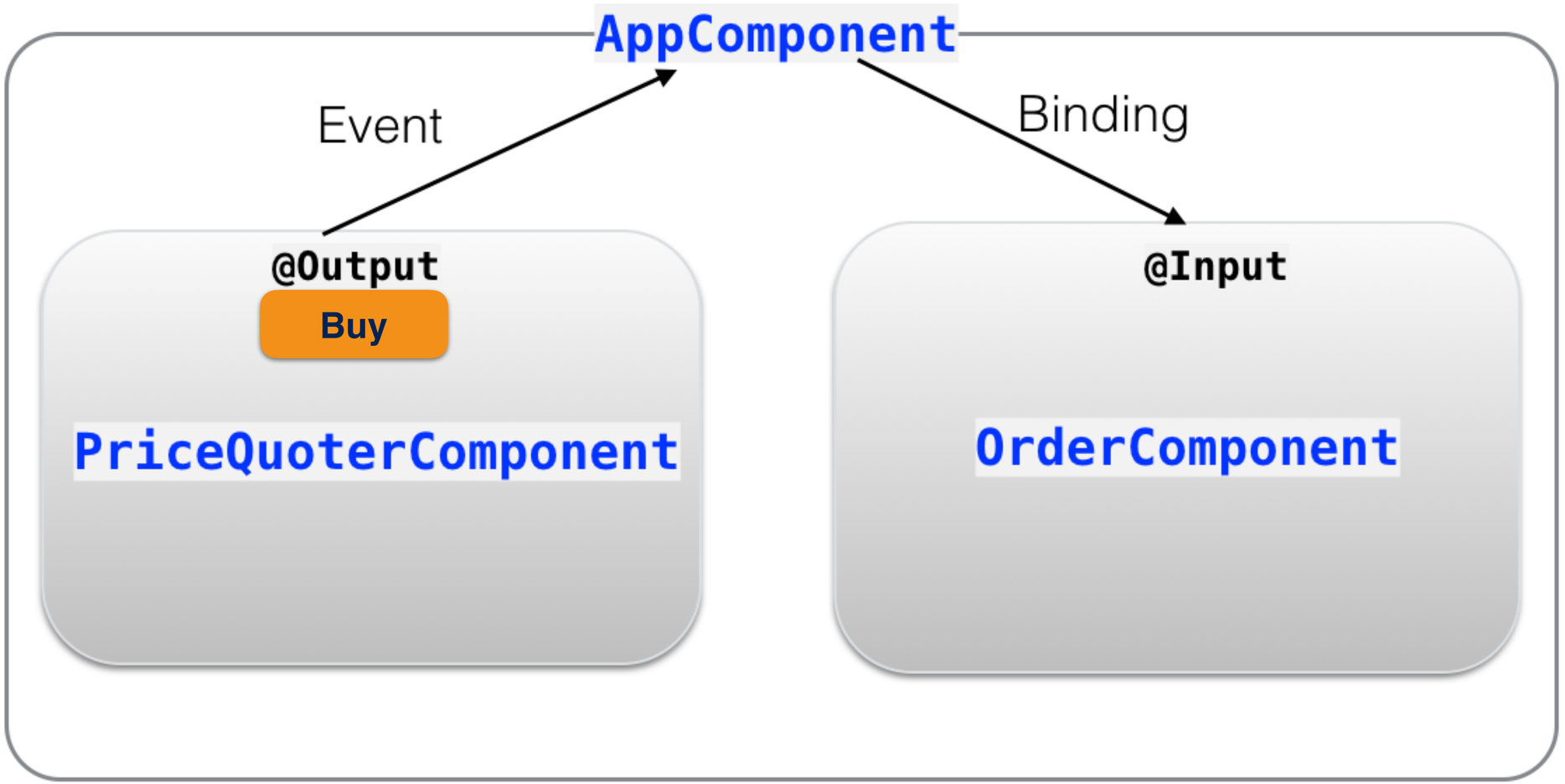
```
@Component({
    selector: 'app',
    template: `
      <price-quoter (lastPrice)="priceQuoteHandler($event)"></price-quoter><br>

      AppComponent received: {{stockSymbol}} {{price | currency:'USD':true}}
    `
})
class AppComponent {

    stockSymbol: string;
    price:number;

    priceQuoteHandler(event:IPriceQuote) {
        this.stockSymbol = event.stockSymbol;
        this.price = event.latestPrice;
    }
}
```

Parent components as mediators

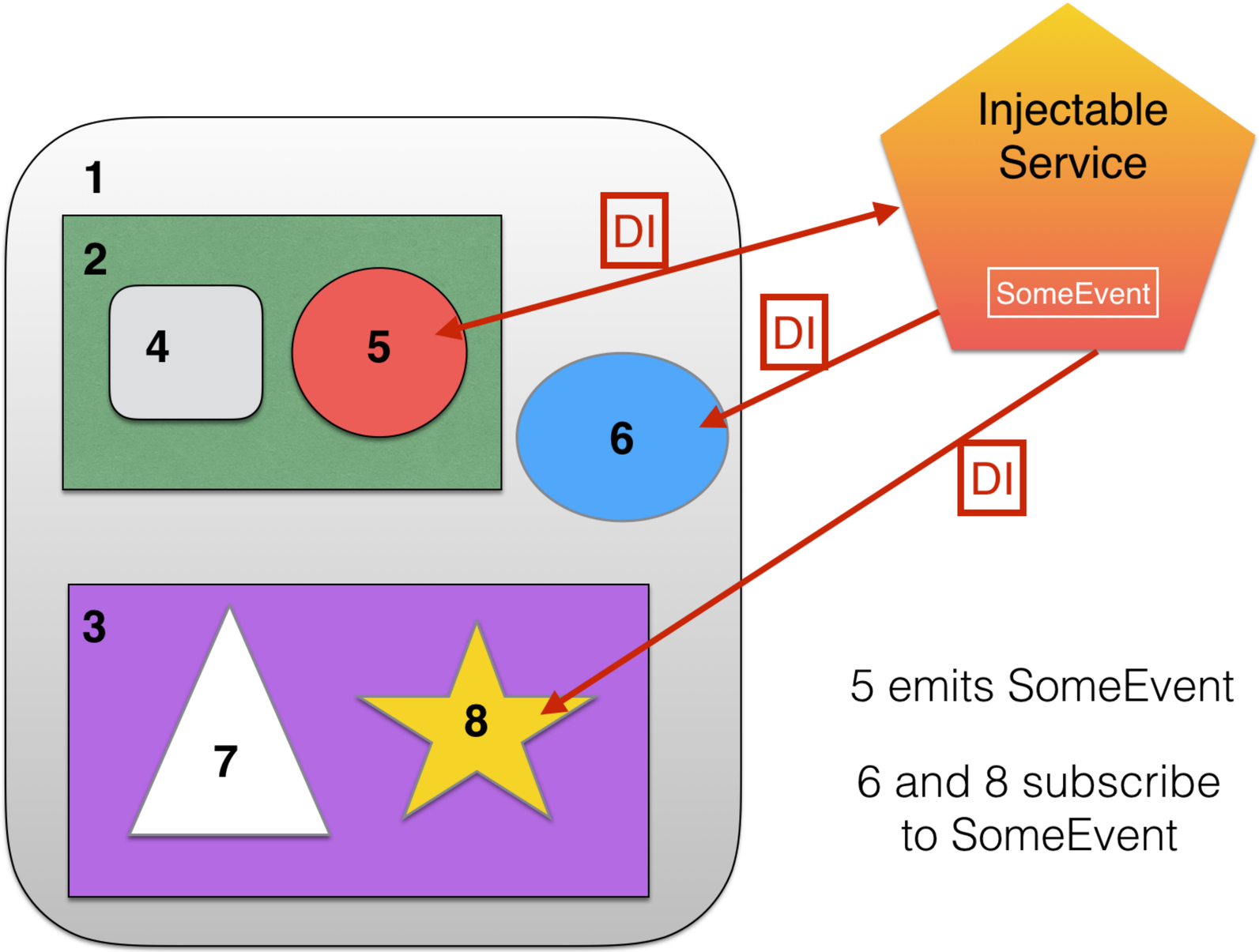


QUIZ: How the number 7 can send the data to number 6?

Parent as a mediator



Injectable service as a mediator

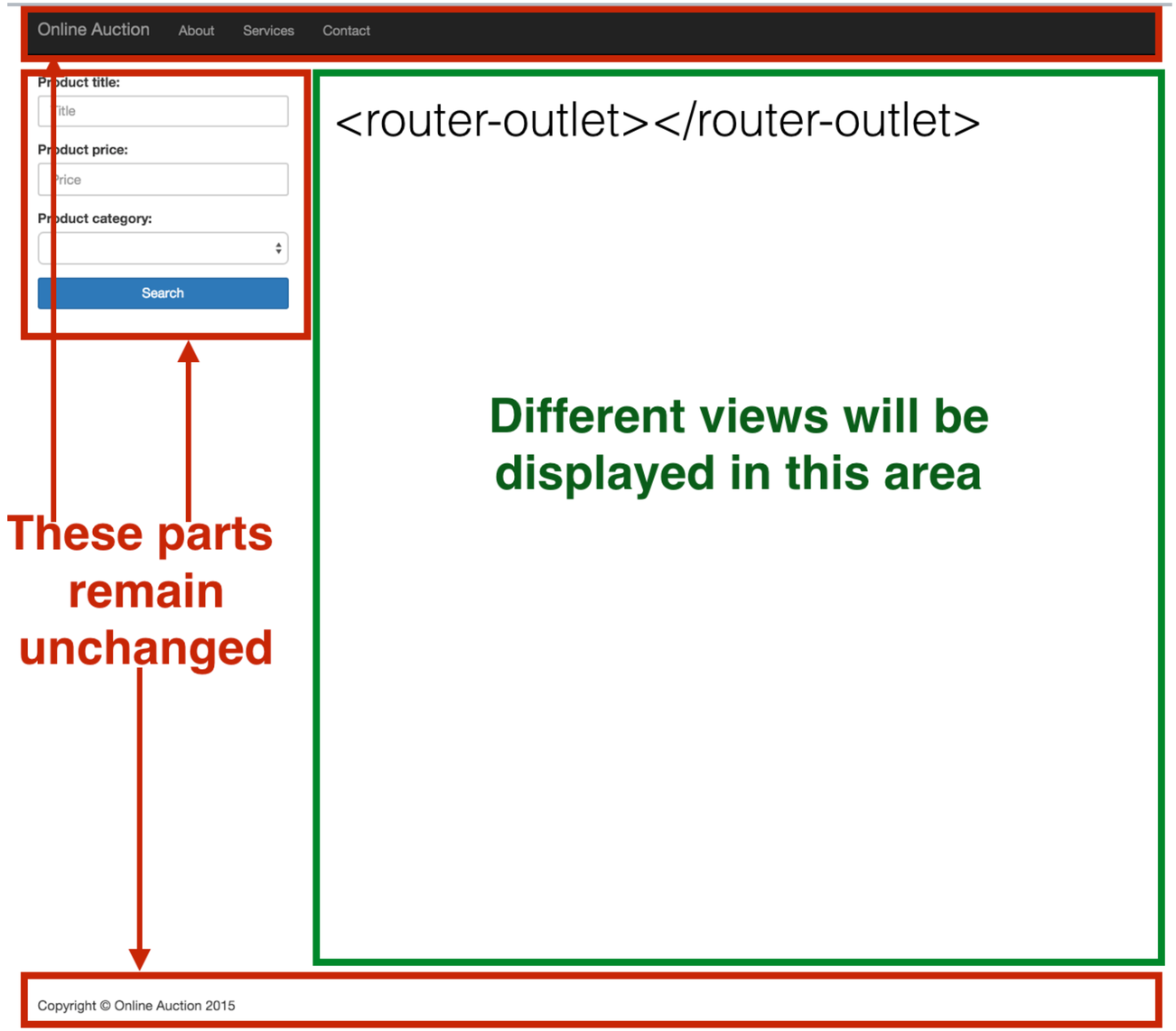


5 emits SomeEvent

6 and 8 subscribe
to SomeEvent

Navigation with Router

Router Outlet



Sample routes configuration

app.routing.ts:

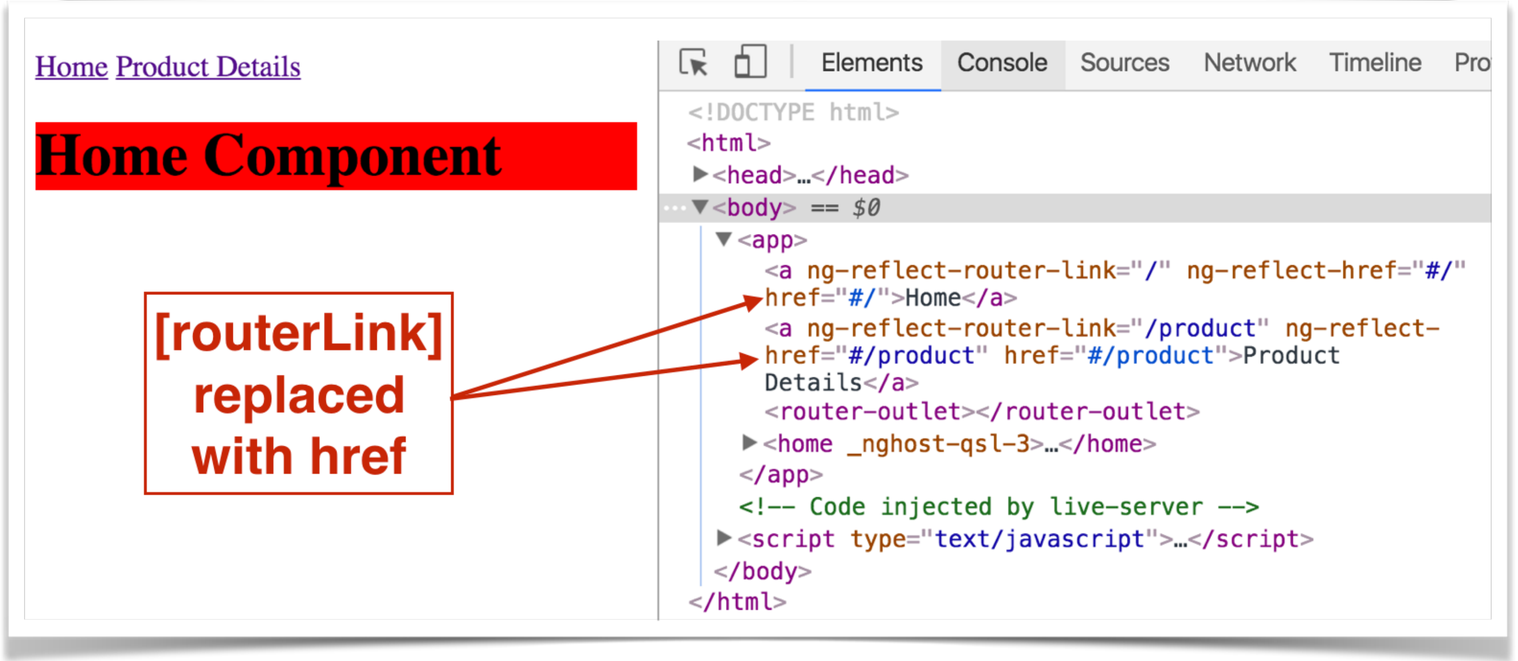
```
import { Routes, RouterModule } from '@angular/router';
import {HomeComponent} from "./home";
import {ProductDetailComponent} from "./product";

const routes: Routes = [
    {path: '',        component: HomeComponent},
    {path: 'product', component: ProductDetailComponent}
];

export const routing = RouterModule.forRoot(routes); // config for the root module
```

The root module

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }  from './components/app.component';
import {HomeComponent} from "./components/home";
import {ProductDetailComponent} from "./components/product";
import {LocationStrategy, HashLocationStrategy} from '@angular/common';

import {routing} from './components/app.routing';

@NgModule({
    imports:      [ BrowserModule, routing ],
    declarations: [ AppComponent, HomeComponent, ProductDetailComponent],

    providers:[{provide: LocationStrategy, useClass: HashLocationStrategy}],

    bootstrap:    [ AppComponent ]
})
class AppModule { }
```

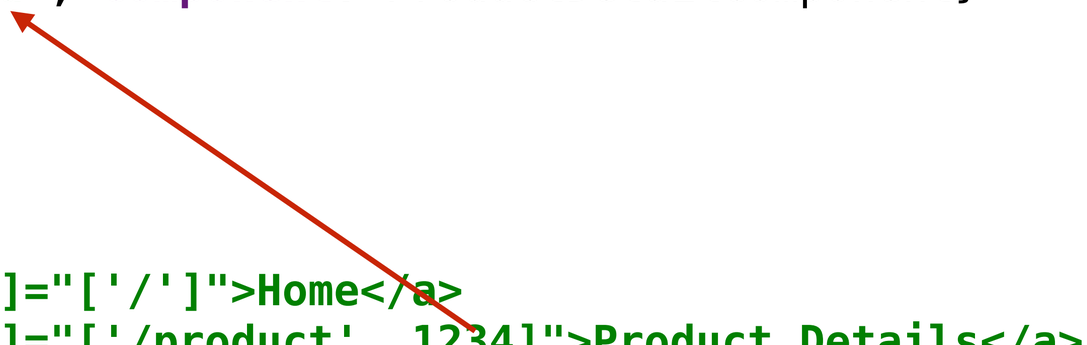
Navigation using template links

```
import {Component} from '@angular/core';
@Component({
    selector: 'app',
    template: `
        <a [routerLink]="['/']">Home</a>
        <a [routerLink]="['/product']">Product Details</a>

        <router-outlet></router-outlet>
    `
})
export class AppComponent {}
```



Passing parameters to a route

```
const routes: Routes = [
    {path: '',          component: HomeComponent},
    {path: 'product/:id', component: ProductDetailComponent}
];

@Component({
    selector: 'app',
    template: `
        <a [routerLink]="['/']">Home</a>
        <a [routerLink]="['/product', 1234]">Product Details</a>
        <router-outlet></router-outlet>
    `
})
class AppComponent {}
```



Receiving params: ActivatedRoute

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

@Component({
  selector: 'product',
  template: `<h1 class="product">Details for product {{productID}}</
h1>`,
  styles: ['.product {background: cyan}']
})
export class ProductDetailComponent {
  productID: string;

  constructor(route: ActivatedRoute) {
    this.productID = route.snapshot.paramMap.get('id');

  }
}
```

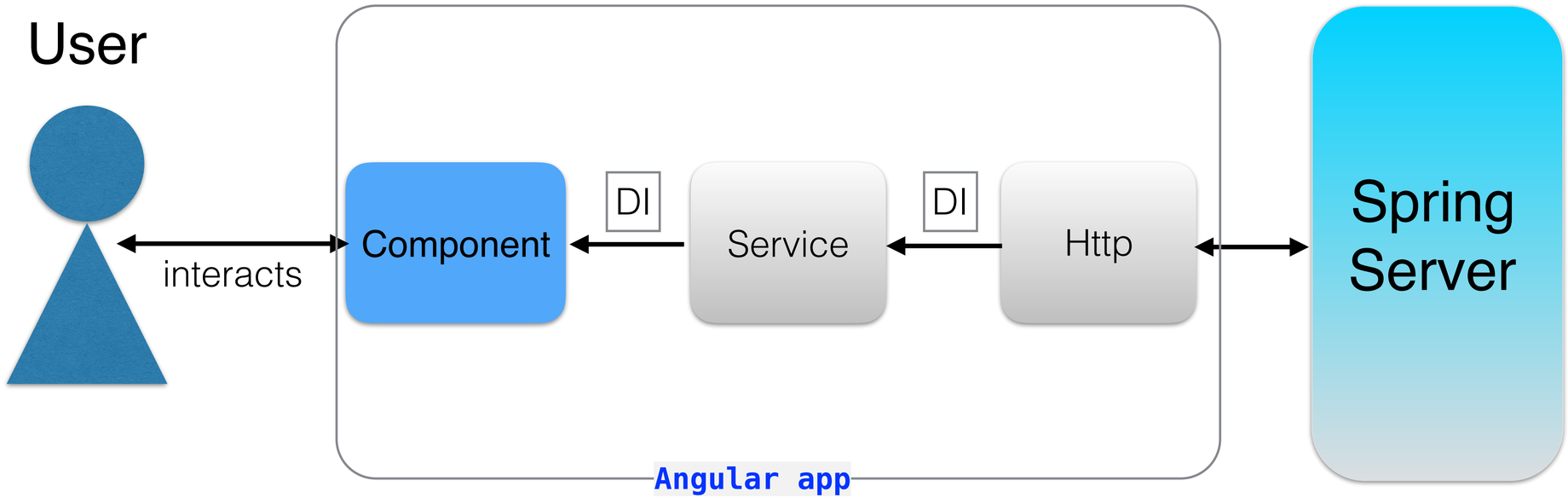
Hands-on

Implementing navigation to the product detail view

(steps 1.8 - 1.10)

Communication with Web server and deployment

Typical workflow



Spring can be replaced with Java EE, NodeJS, .Net, or other technology.

Angular HttpClient

Declare `HttpClientModule` in `@NgModule` and inject an `HttpClient` object into the constructor of a service (or component)

```
import {HttpClientModule}
              from ‘@angular/common/http’;
…
@NgModule({
    imports:      [ BrowserModule,
                    HttpClientModule],
    declarations: [ AppComponent],
    bootstrap:    [ AppComponent ]
})
class AppModule { }
```

```
import {HttpClient} from '@angular/http';

class ProductService {

 constructor(private httpClient: HttpClient){}

 getProducts(){
  this.httpClient.get('products').subscribe(...);
 }
}
```

Server endpoint

Subscribing to HttpClient

```
@Component({
  selector: 'app-root',
  template: `<h1>All Products</h1>
  <ul>
    <li *ngFor="let product of products">   ←3
       {{product.title}} {{product.price}}
    </li>
  </ul>
  {{error}}
  `})
export class AppComponent implements OnInit, OnDestroy{

  products: any[] = [];
  theDataSource: Observable<any[]>;
  productSubscription: Subscription;
  error:string;

  constructor(private httpClient: HttpClient) {
    this.theDataSource = this.httpClient.get<any[]>('/api/products');  ←1
  }

  ngOnInit(){
    this.productSubscription = this.theDataSource
      .subscribe(
      data => {
          this.products=data;      ←2
      },
      err =>
        this.error = `Can't get products. Got ${err.status} from ${err.url}`
    );
  }

  ngOnDestroy(){
    this.productSubscription.unsubscribe();  ←4
  }
}
```

Building apps for prod deployment

JiT vs AoT compilation

- **Just-in-Time**: your app includes Angular's compiler and is dynamically compiled in the browser.
- **Ahead-of-Time**: Angular components and templates are precompiled into JS with the **ngc** compiler.
- The AoT-compiled apps don't include the Angular compiler

ng build

the output goes into the **dist** dir

- ng build ← non-optimized build, produces files
- ng build -prod ← optimized build, produces files
- ng serve -prod ← optimized build, in-memory bundles

Sample npm scripts for deployment on Spring Boot server

```
"scripts": {

  "build": "ng build",

  "postbuild": "npm run deploy",

  "predeploy": "rimraf ../server/src/main/resources/static
                && mkdirp ../server/src/main/resources/static",

  "deploy": "copyfiles -f dist/** ../server/src/main/resources/static"
}
```

post runs after

pre runs before

static resources

Install these dependencies:
npm i rimraf copyfiles mkdirp --save-dev

Demo
myStore and Spring Boot server

Thank you!

- Training inquiries:
  training@faratasystems.com
- My blog:
  yakovfain.com
- Twitter: @yfain