More on Components



Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/







Improving Our Components

Strong typing & interfaces

Encapsulating styles

Lifecycle hooks

Custom pipes

Nested components



Module Overview



Defining an Interface

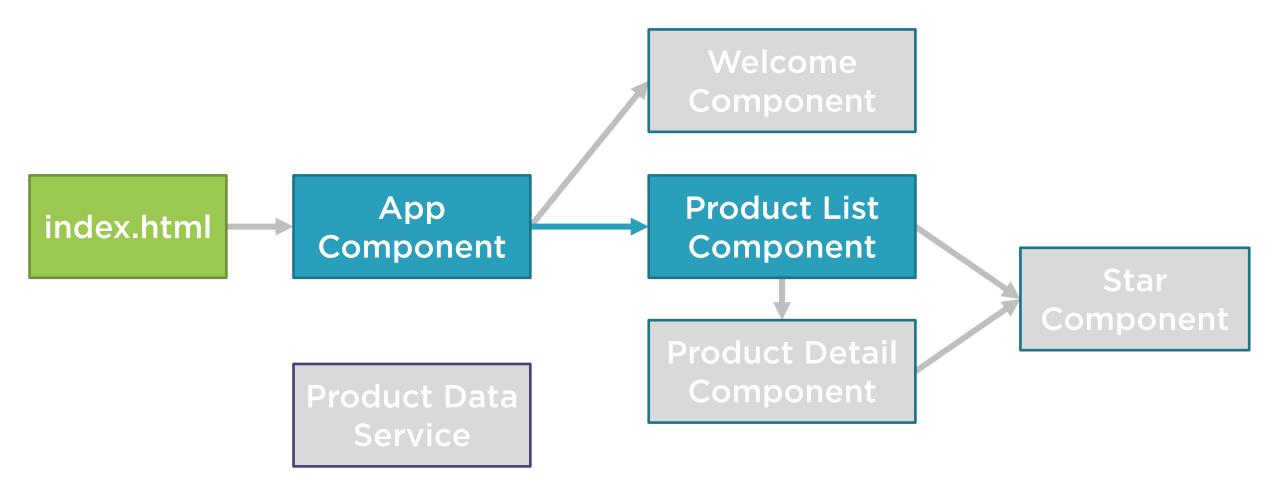
Encapsulating Component Styles

Using Lifecycle Hooks

Building a Custom Pipe



Application Architecture





Strong Typing

```
export class ProductListComponent {
   pageTitle: string = 'Product List';
   showImage: boolean = false;
   listFilter: string = 'cart';
   message: string;
   products: any[] = [...];
   toggleImage(): void {
      this.showImage = !this.showImage;
   onRatingClicked(message: string): void {
        this.message = message;
```



Interface

A specification identifying a related set of properties and methods.

A class commits to supporting the specification by implementing the interface.

Use the interface as a data type.

Development time only!



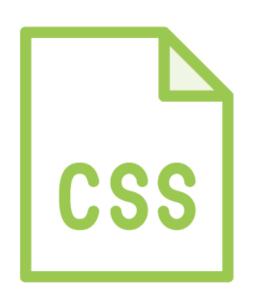
Interface Is a Specification

```
export interface IProduct {
                                                 export
    productId: number;
                                                keyword
    productName: string;
    productCode: string;
                                                Interface
    releaseDate: Date;
                                                 Name
    price: number;
    description: string;
                                               interface
    starRating: number;
                                                keyword
    imageUrl: string;
    calculateDiscount(percent: number): number;
```

Using an Interface as a Data Type

```
import { IProduct } from './product';
export class ProductListComponent {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';
  products: IProduct[] = [...];
  toggleImage(): void {
      this.showImage = !this.showImage;
```

Handling Unique Component Styles



Templates sometimes require unique styles

We can inline the styles directly into the HTML

We can build an external stylesheet and link it in index.html

There is a better way!



Encapsulating Component Styles

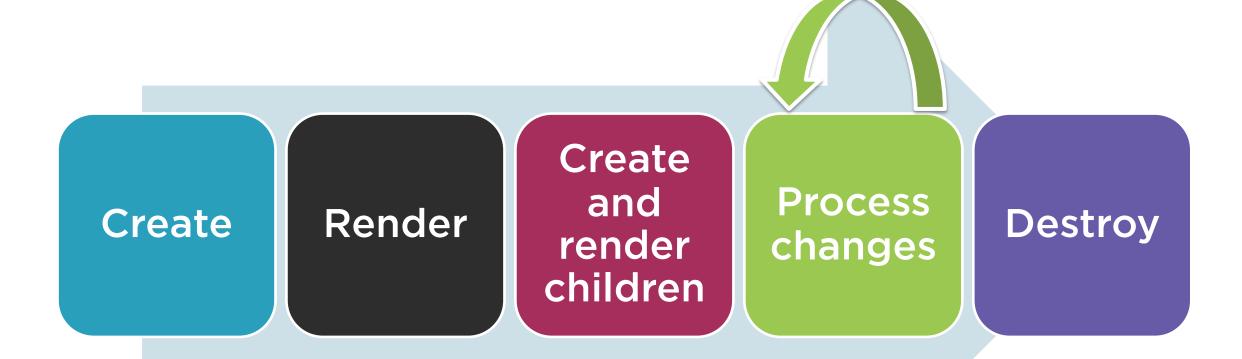
styles

```
@Component({
    selector: 'pm-products',
    templateUrl: './product-list.component.html',
    styles: ['thead {color: #337AB7;}']})
```

styleUrls

```
@Component({
    selector: 'pm-products',
    templateUrl: './product-list.component.html',
    styleUrls: ['./product-list.component.css']})
```

Component Lifecycle



Component Lifecycle Hooks



Onlnit: Perform component initialization, retrieve data

OnChanges: Perform action after change to input properties

OnDestroy: Perform cleanup



Using a Lifecycle Hook





Transforming Data with Pipes

Transform bound properties before display

Built-in pipes

- date
- number, decimal, percent, currency
- json, slice
- etc

Custom pipes



Building a Custom Pipe

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
    name: 'convertToSpaces'
export class ConvertToSpacesPipe
                  implements PipeTransform {
  transform(value: string,
            character: string): string {
```

Using a Custom Pipe

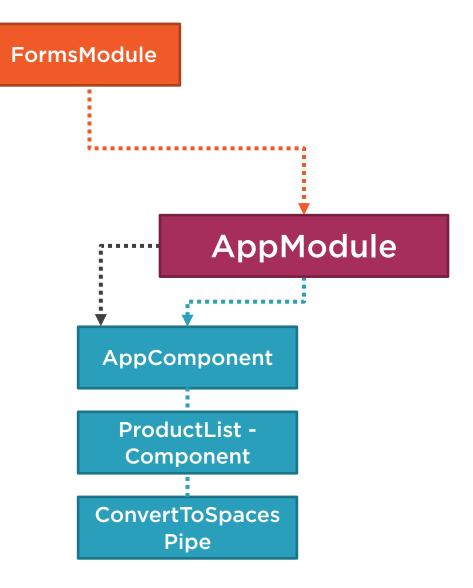
Template

```
{{ product.productCode | convertToSpaces:'-'}}
```

Pipe

```
transform(value: string, character: string): string {
}
```







..... Exports

•• Declarations

BrowserModule

Providers

Bootstrap



Using a Custom Pipe

Template

```
{{ product.productCode | convertToSpaces:'-'}}
```

Module

```
@NgModule({
  imports: [
      BrowserModule,
      FormsModule ],
  declarations: [
      AppComponent,
      ProductListComponent,
      ConvertToSpacesPipe ],
  bootstrap: [ AppComponent ]
export class AppModule { }
```

Filtering Data



"Angular doesn't offer such pipes because they perform poorly and prevent aggressive minification."

angular.io



"The Angular team and many experienced Angular developers strongly recommend moving filtering and sorting logic into the component itself."

angular.io



Getters and Setters

```
listFilter: string = 'cart';
```

```
private _listFilter: string;
get listFilter(): string {
   return this._listFilter;
}
set listFilter(value: string) {
   this._listFilter = value;
}
```

Getter

Defines a read-only property

```
get fullName(): string {
  return this.lastName + ', ' + this.firstName;
}
```

```
get showImage(): boolean {
  return this.productParameterService.showImage;
}
```

```
console.log(this.fullName);
```



Setter

Defines a write-only property

```
set quantity(value: number) {
   this.recalculate(quantity);
}
```

```
this.quantity = 10;
```

```
private _listFilter = '';
set listFilter(value: string) {
   this._listFilter = value;
   this.filteredProducts = this.performFilter(value);
}
```

Getters and Setters

```
private _listFilter: string;
get listFilter(): string {
   return this._listFilter;
}
set listFilter(value: string) {
   this._listFilter = value;
}
```



Checklist: Interfaces



Defines custom types

Creating interfaces:

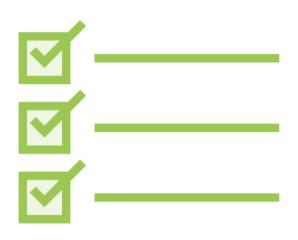
- interface keyword
- export it

Implementing interfaces:

- implements keyword & interface name
- Write code for each property & method



Checklist: Encapsulating Styles



styles property

- Specify an array of style strings

styleUrls property

- Specify an array of stylesheet paths



Checklist: Using Lifecycle Hooks



Import the lifecycle hook interface
Implement the lifecycle hook interface
Write code for the hook method



Checklist: Building a Custom Pipe



Import Pipe and PipeTransform

Create a class that implements PipeTransform

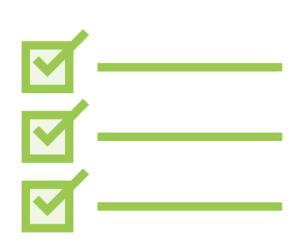
- **export** the class

Write code for the Transform method

Decorate the class with the Pipe decorator



Checklist: Using a Custom Pipe



Import the custom pipe

Add the pipe to the declarations array of an Angular module

Any template associated with a component that is also declared in that Angular module can use that pipe

Use the Pipe in the template

- Pipe character
- Pipe name
- Pipe arguments (separated with colons)



Summary



Defining an Interface

Encapsulating Component Styles

Using Lifecycle Hooks

Building a Custom Pipe



Application Architecture

