

More on Components

Introduction



Deborah Kurata

Consultant | Speaker | Author | MVP | GDE

@deborahkurata



Improving Our Components



Strong typing & interfaces



Encapsulating styles



Lifecycle hooks



Custom pipes



Nested components

Module Overview



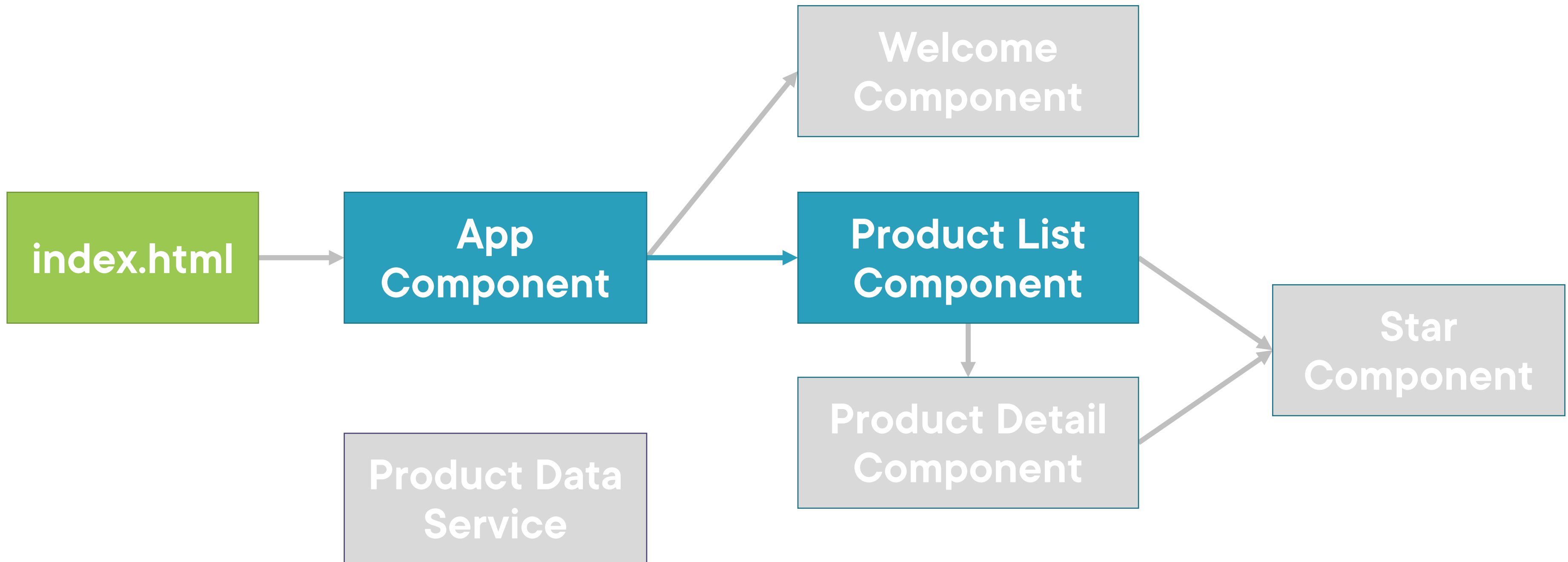
Defining an interface

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;  
  }  
}
```


An **interface** is a **specification**
identifying a related set of
properties and methods.

Two Ways to Use an Interface

```
export interface IProduct {  
  productId: number;  
  productName: string;  
  productCode: string;  
  releaseDate: string;  
  price: number;  
  description: string;  
  starRating: number;  
  imageUrl: string;  
}
```

As a type

```
products: IProduct[] = [];
```

```
export interface DoTiming {  
  count: number;  
  start(index: number): void;  
  stop(): void;  
}
```

As a feature set

```
export class myComponent  
  implements DoTiming {  
  count: number = 0;  
  start(index: number): void {  
    ...  
  }  
  stop(): void {  
    ...  
  }  
}
```


Declaring an Interface as a Data Type

```
export interface IProduct {  
    productId: number;  
    productName: string;  
    productCode: string;  
    releaseDate: Date;  
    price: number;  
    description: string;  
    starRating: number;  
    imageUrl: string;  
}
```

**export
keyword**

**Interface
name**

**interface
keyword**

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

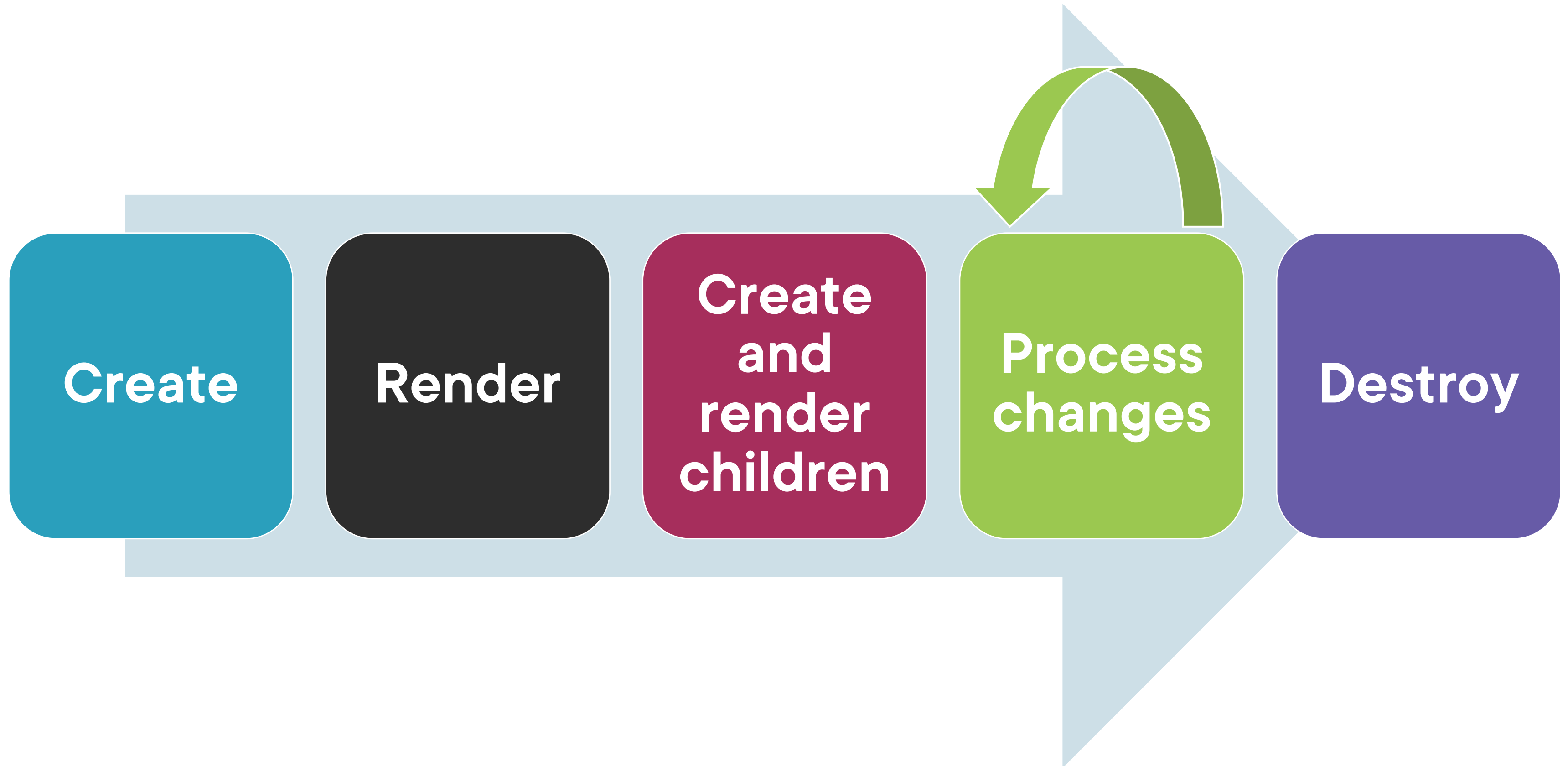
styles

```
@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



A lifecycle hook is an interface we implement to write code when a component lifecycle event occurs.

Component Lifecycle Hooks



OnInit: Perform component initialization, retrieve data

OnChange: Perform action after change to input properties

OnDestroy: Perform cleanup

Using a Lifecycle Hook

2

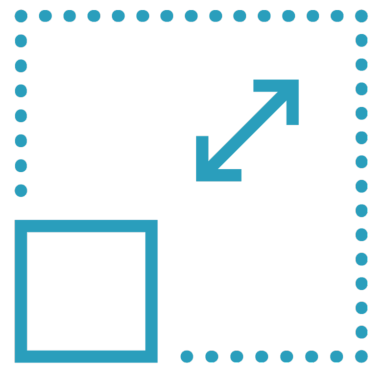
1

```
export class ProductListComponent implements OnInit {  
  pageTitle: string = 'Product List';  
  showImage: boolean = false;  
  listFilter: string = 'cart';  
  products: IProduct[] = [...];
```

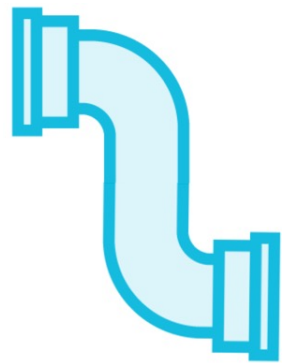
3

```
}
```

Transforming Data with Pipes



Transform bound properties before display



Built-in pipes: date, number, decimal, percent, currency, json, 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 {
  }
}
```

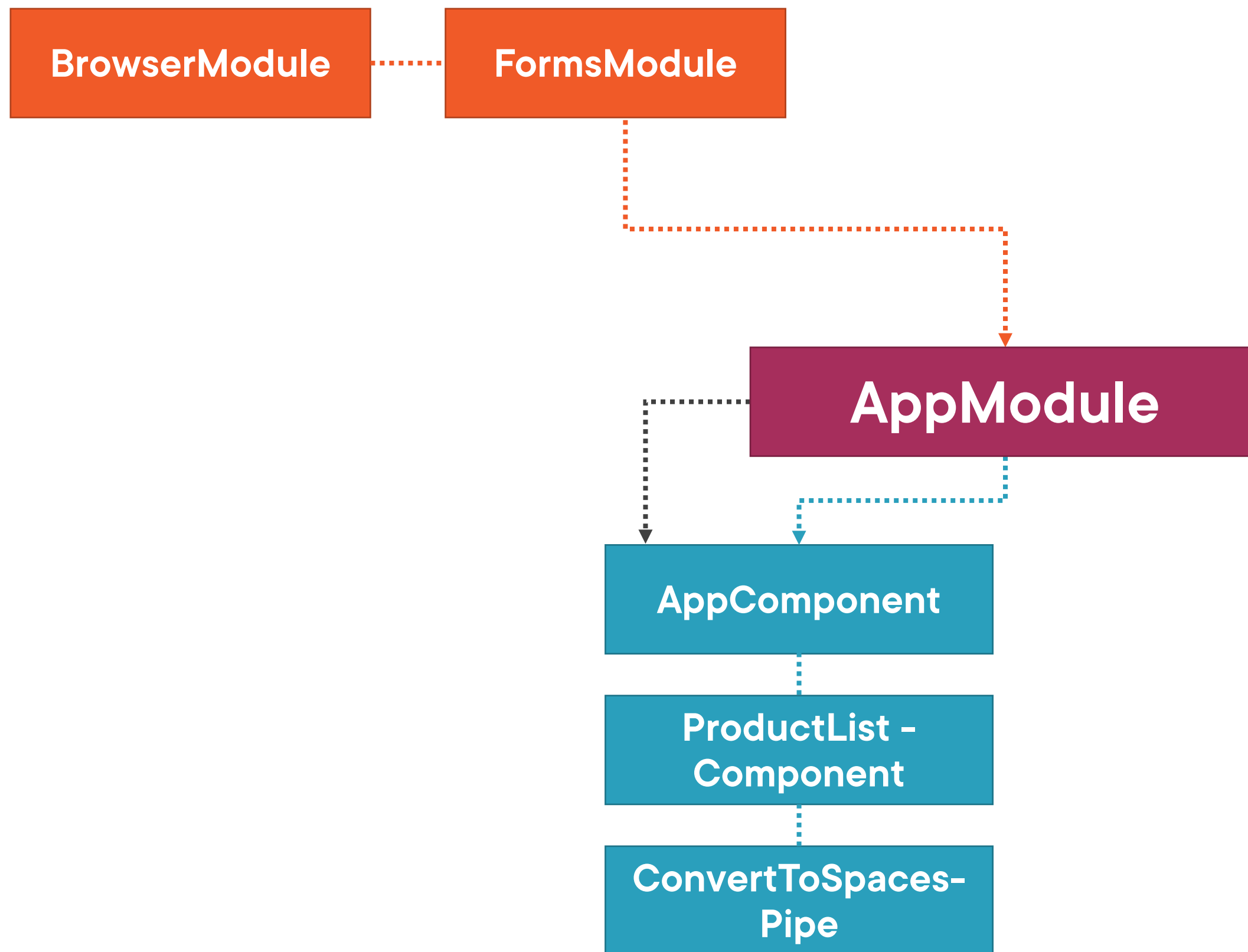
Using a Custom Pipe

Template

```
<td>{{ product.productCode | convertToSpaces:'-' }}</td>
```

Pipe

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



- Imports
- Exports
- Declarations
- Bootstrap

Using a Custom Pipe

Template

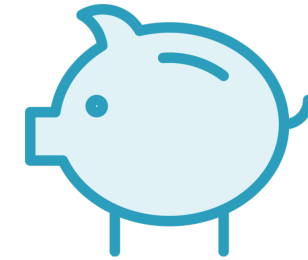
```
<td>{{ product.productCode | convertToSpaces:'-' }}</td>
```

Module

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

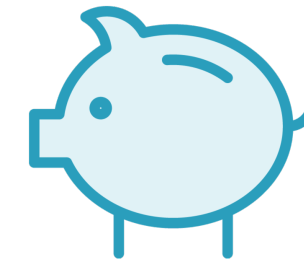
Getters and Setters

```
amount: number = 0;
```



```
get amount(): number {  
    // process the amount  
    // return amount from private storage  
}  
set amount(value: number) {  
    // process the amount  
    // retain amount in private storage  
}
```


Getters and Setters



```
amount: number = 0;
```

```
private _amount: number = 0;
```

```
get amount(): number {  
    // process the amount  
    // return amount from private storage  
    return this._amount;  
}  
set amount(value: number) {  
    // process the amount  
    // retain amount in private storage  
    this._amount = value;  
}
```

Getters and Setters

```
private _amount: number = 0;
```

```
get amount(): number {  
    // process the amount  
    // return amount from private storage  
    return this._amount;  
}  
set amount(value: number) {  
    // process the amount  
    // retain amount in private storage  
    this._amount = value;  
}
```

```
this.amount = 200;
```

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

Filtering a List

```
products: IProduct[] = [...];
```

```
performFilter(): IProduct[] {  
    return this.products.filter();  
}
```

An **arrow function** is compact syntax for defining a function.

Filtering a List

```
products: IProduct[] = [...];
```

```
performFilter(): IProduct[] {  
    return this.products.filter((product: IProduct) =>  
        product.productName.includes(this.listFilter));  
}
```

Arrow Functions

Classic named function (method)

```
capitalizeName(product: IProduct): string {  
    return product.productName.toUpperCase();  
}
```

Arrow function

```
(product: IProduct) => product.productName.toUpperCase();
```

Multi-statement arrow function

```
(product: IProduct) => {  
    console.log(product.productName);  
    return product.productName.toUpperCase();  
}
```

Interface Checklist:

Interface as a Type



interface keyword

Properties and their types

Export it

```
export interface IProduct {  
    productId: number;  
    productName: string;  
    productCode: string;  
    ...  
}
```

Use the interface as a data type

```
products: IProduct[] = [...];
```


Interface Checklist:

Interface as a Feature Set



Implementing interfaces:

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

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

export class ProductComponent implements OnInit {
  ngOnInit(): void {
    console.log('In OnInit');
  }
}
```

Styles Checklist: Encapsulating Styles



styles property

- Specify an array of style strings

styleUrls property

- Specify an array of stylesheet paths

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

Lifecycle Hook Checklist:

Using Lifecycle Hooks



Import the lifecycle hook interface

Implement the lifecycle hook interface

Write code for the hook method

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

export class ProductComponent implements OnInit {
  ngOnInit(): void {
    console.log('In OnInit');
  }
}
```

Pipe Checklist: Building a Custom Pipe



Create a class that implements PipeTransform

Write code for the Transform method

Decorate the class with the Pipe decorator

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

@Pipe({
  name: 'spacePipe'
})
export class SpacePipe implements PipeTransform {
  transform(value: string,
            character: string): string { ... }
}
```

Pipe Checklist: Using a Custom Pipe



Add the pipe to the declarations array of an Angular module

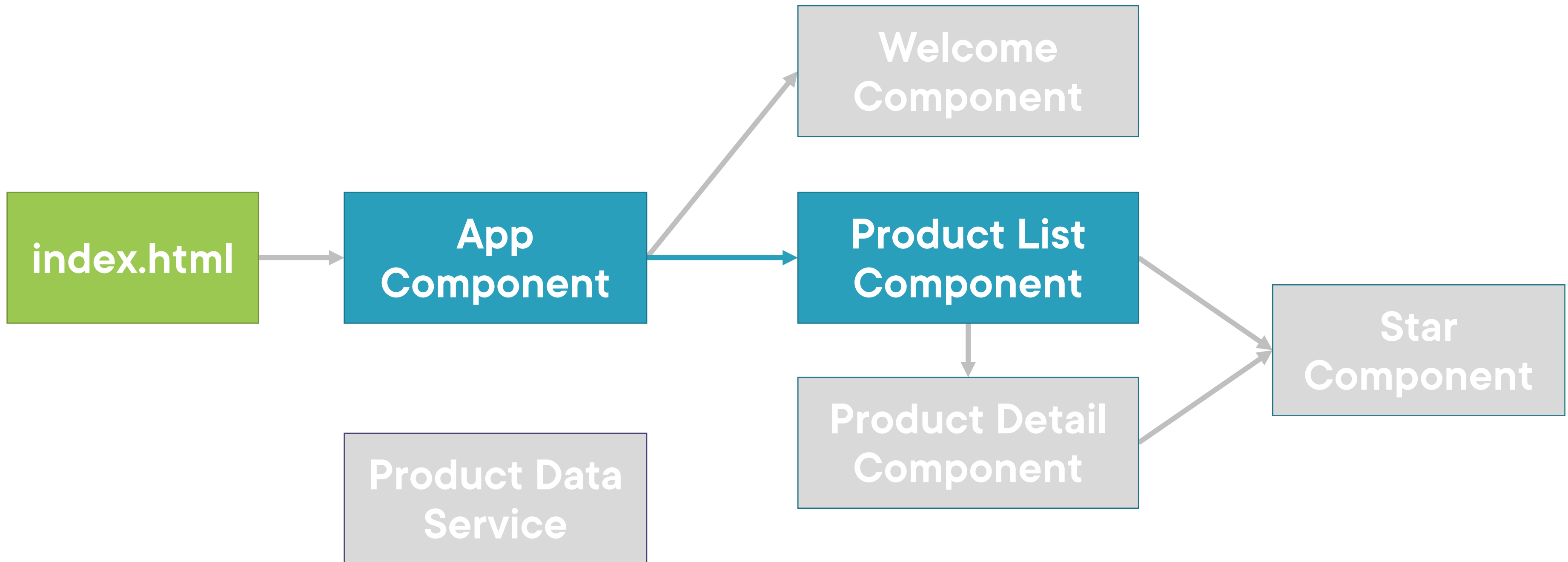
```
@NgModule({  
  imports: [...],  
  declarations: [  
    AppComponent,  
    ProductListComponent,  
    SpacePipe ],  
  bootstrap: [ AppComponent ]  
})  
export class AppModule { }
```

Use the pipe in a template

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

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

Application Architecture





Coming up next ...

Building Nested Components

Product List					
Filter by:		<input type="text"/>			
Show Image	Product	Code	Available	Price	5 Star Rating
	Leaf Rake	gdn 0011	March 19, 2021	\$19.95	★★★★
	Garden Cart	gdn 0023	March 18, 2021	\$32.99	★★★★
	Hammer	tbx 0048	May 21, 2021	\$8.90	★★★★★
	Saw	tbx 0022	May 15, 2021	\$11.55	★★★★
	Video Game Controller	gmg 0042	October 15, 2020	\$35.95	★★★★★