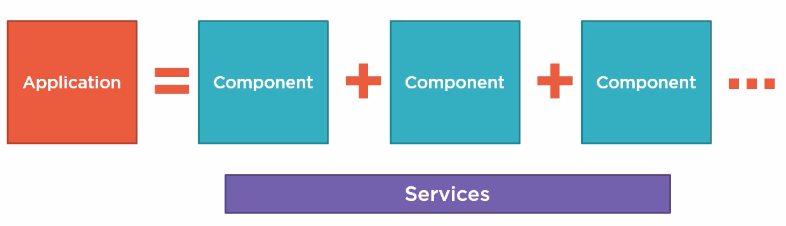
Angular:

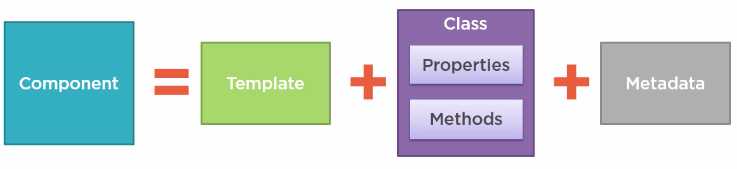
* Data Binding
* Expressive HTML
* Modular
* Built in Service Integration
* Faster rendering time

An angular application consists of components and services.



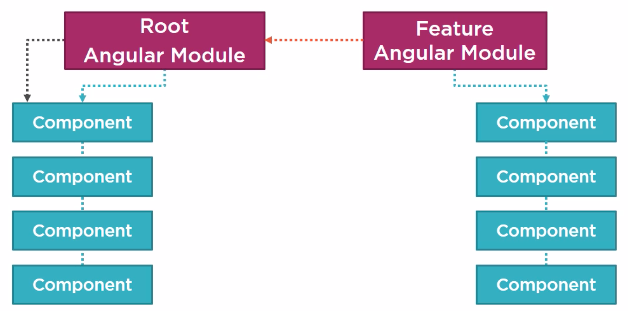
A component has the following parts:

* Template - HTML code or view
* Class – Associated code to support data elements in HTML template also known as properties and methods which respond to button clicks etc.
* Metadata

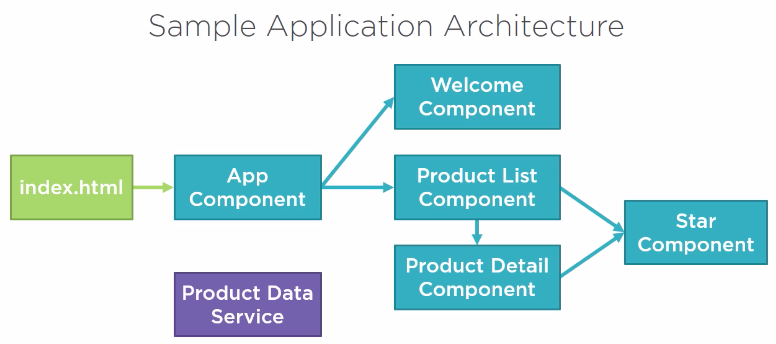


Angular apps are modularized. Each app must have at least one module known as root module.

It can also have feature modules.



<https://github.com/DeborahK/Angular-GettingStarted>



Language: **Typescript**

* Strongly typed (data type present for variables)
* Trans piled to JavaScript.
* Object oriented features

**Set up Angular**

Node Package Manager – Command Line Utility

<https://nodejs.org/en/download/>

This package has installed:

Node.js v10.15.1 to /usr/local/bin/node

npm v6.4.1 to /usr/local/bin/npm

Make sure that /usr/local/bin is in your $PATH.

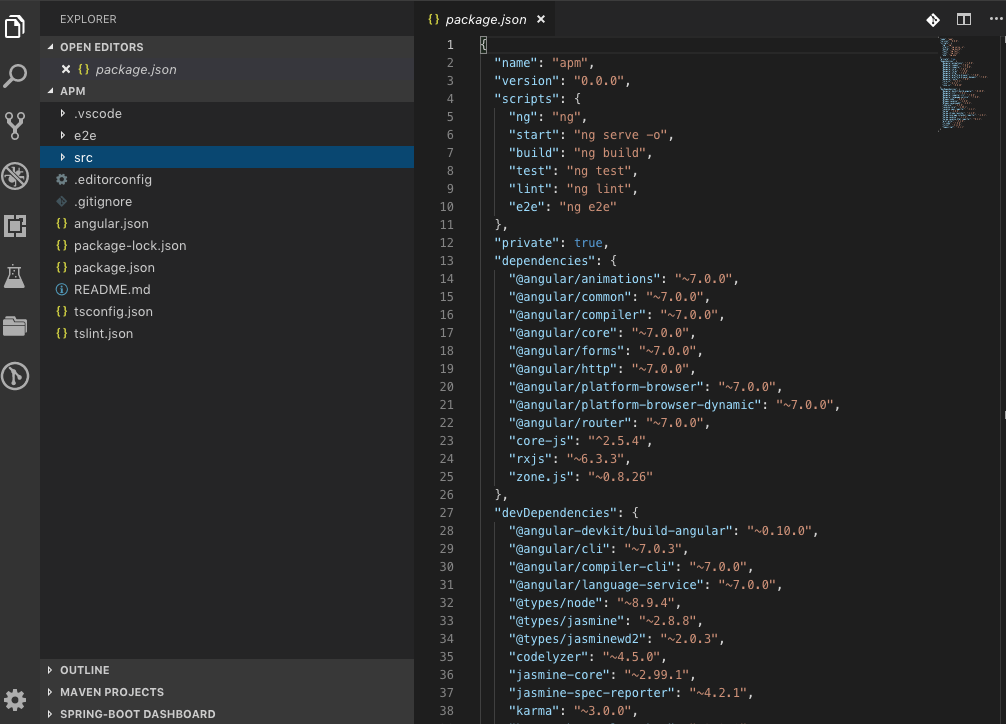
**Steps to setup an angular application:**



Use angular CLI to generate setup and boiler plate code.

<https://github.com/angular/angular-cli>

**An angular Project Sample:**



The code is present under src/

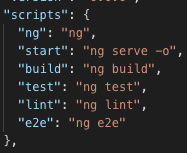
All required libraries are defined in package.json

Navigate to the folder which contains package.json in terminal.

* npm -v to check angular version.
* npm install to install dependencies defined in package.json.

A node\_modules folder is created where all libraries are stored.

**Start Angular App:**



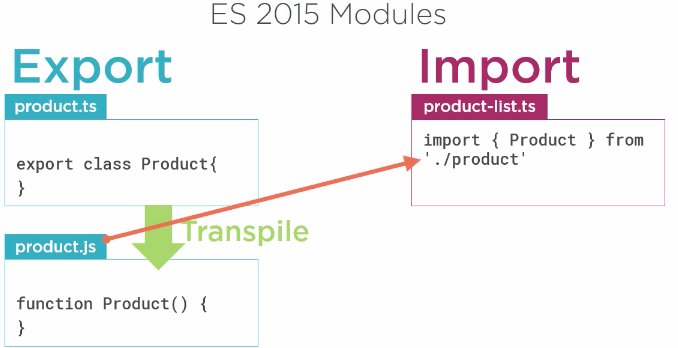
Package.json scripts section has all the start scripts configured.

npm start = ng serve -o (ng = angular, serve = start app, -o = open in default browser)

so npm start can be used to start up our app in default browser.

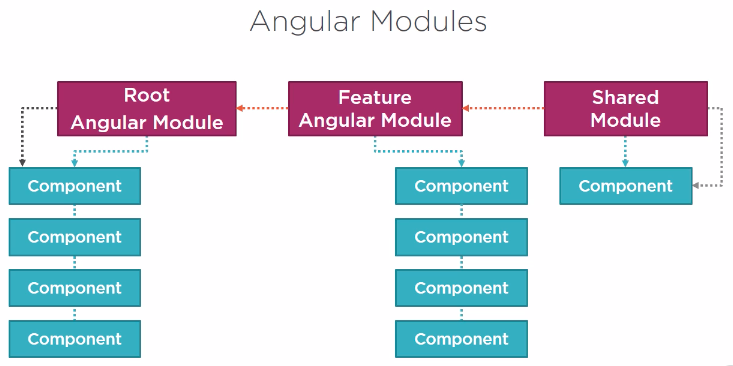
**Modules**

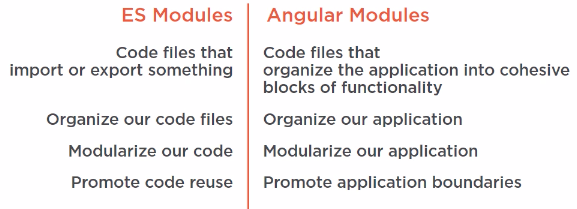
ES Modules:



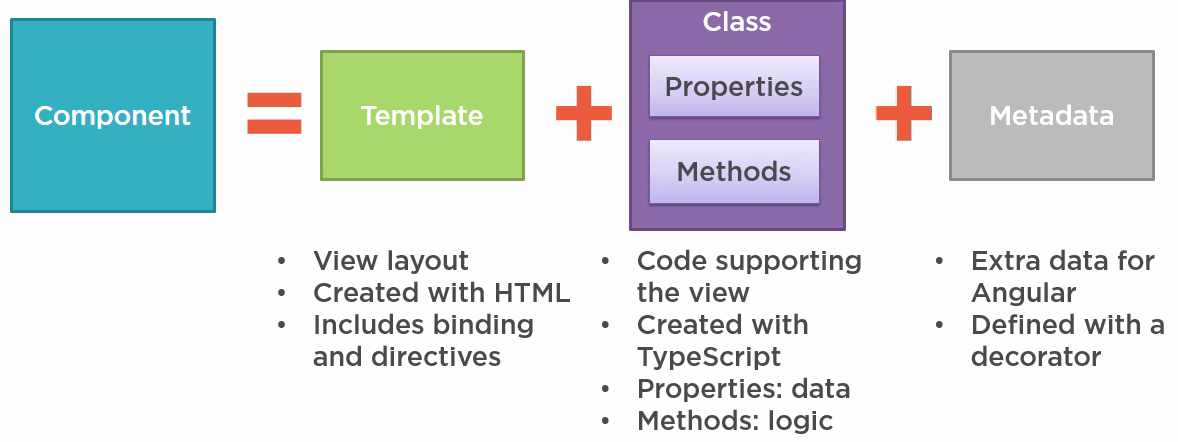
The class Product is exported from product.js (trans piled from .ts) and imported by product-list.ts. Both these files are ES modules.

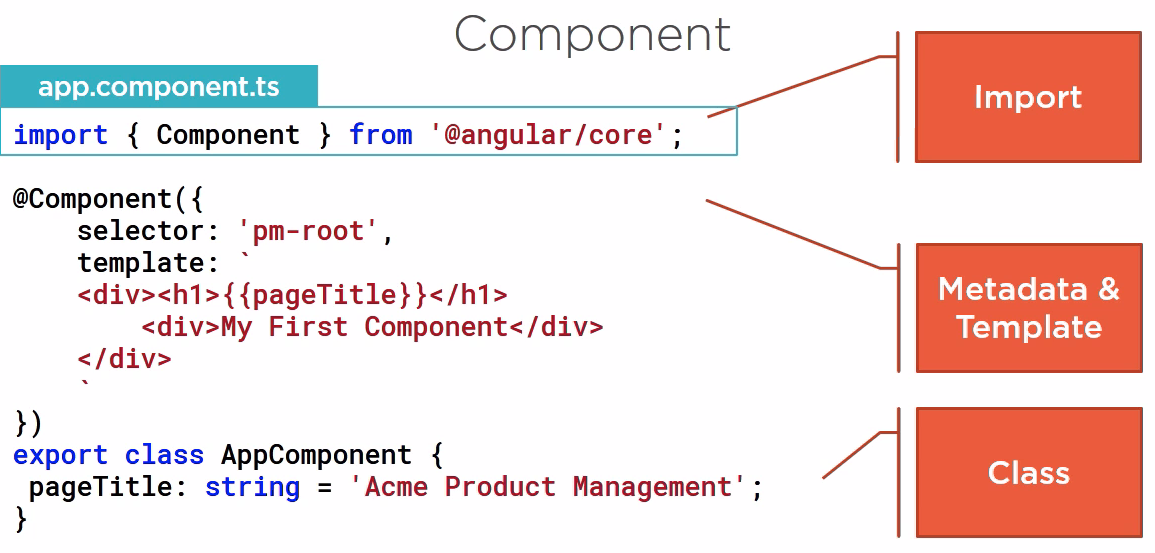
Angular Modules:





**Component:**





The class defines the properties and methods needed by the view.

export class AppComponent {

title = 'Angular: Getting Started';

}

The component decorator defines the metadata which includes template that lays out the view managed by this component

@Component({

selector: 'pm-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

Import imports the members needed.

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

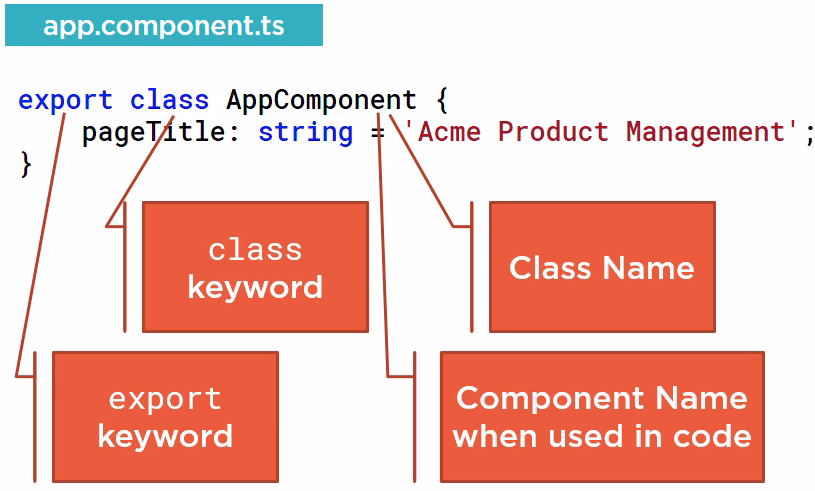
Different parts of a component:

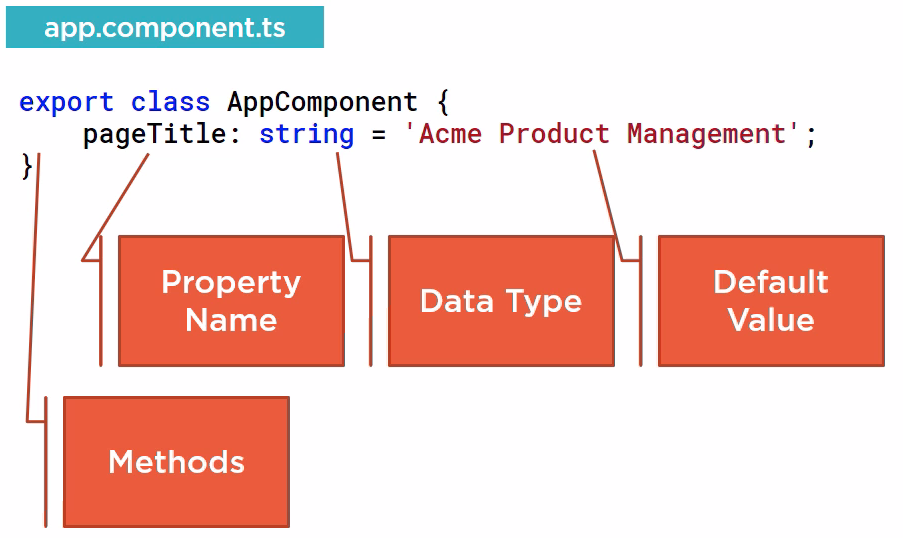
By convention, the root component in Angular is AppComponent.

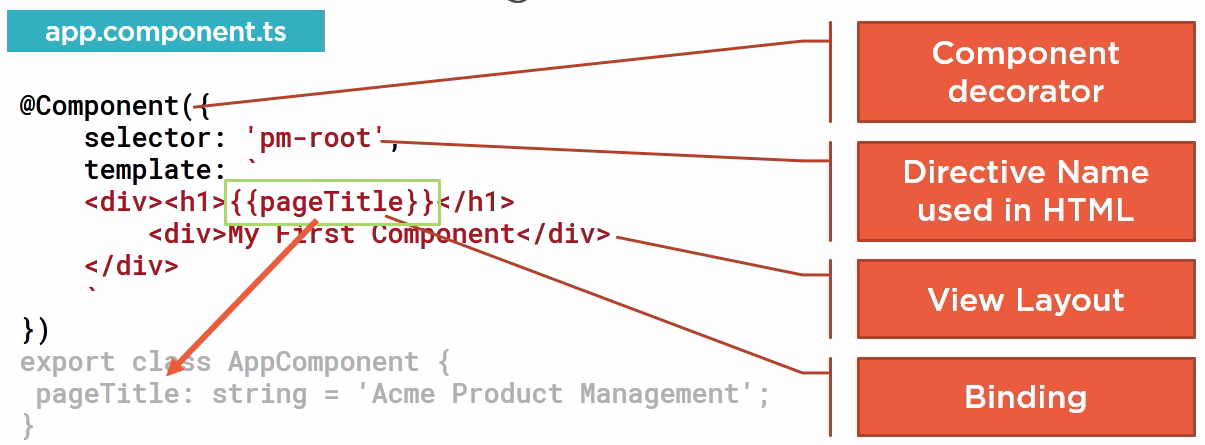
export class AppComponent {

pageTitle: String = 'Acme Product Management';

}







An angular component becomes a component if it is decorated with @Component() decorator.

Decorators are functions which add metadata to the feature they decorate.

@Component() is a decorator.

It accepts and object within curly braces which can have many attributes.

The object here is below.

{

selector: 'pm-root',

template: `

<div>

<h1>{{pageTitle}}</h1>

<div>My First Component</div>

</div>

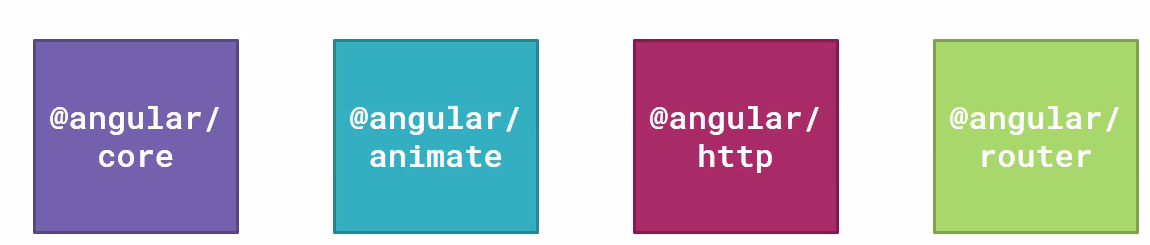
`

}

The selector attribute is used if the template is to be displayed in html using <pm-root></pm-root>.

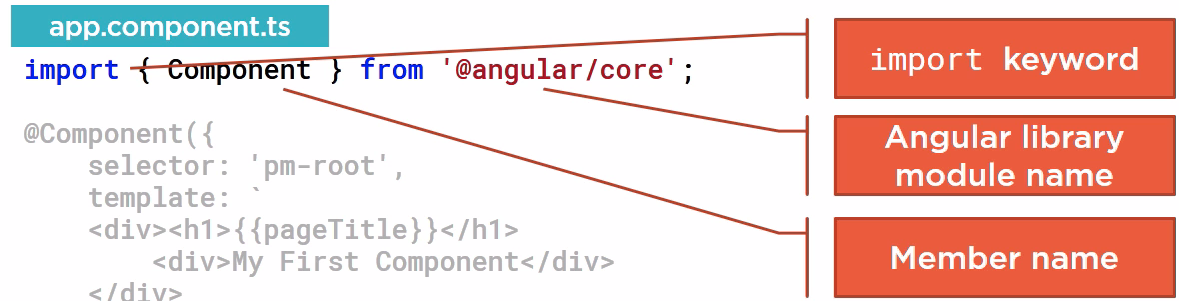
{{pageTitle}} is used to bind to property.

Different Angular Modules:

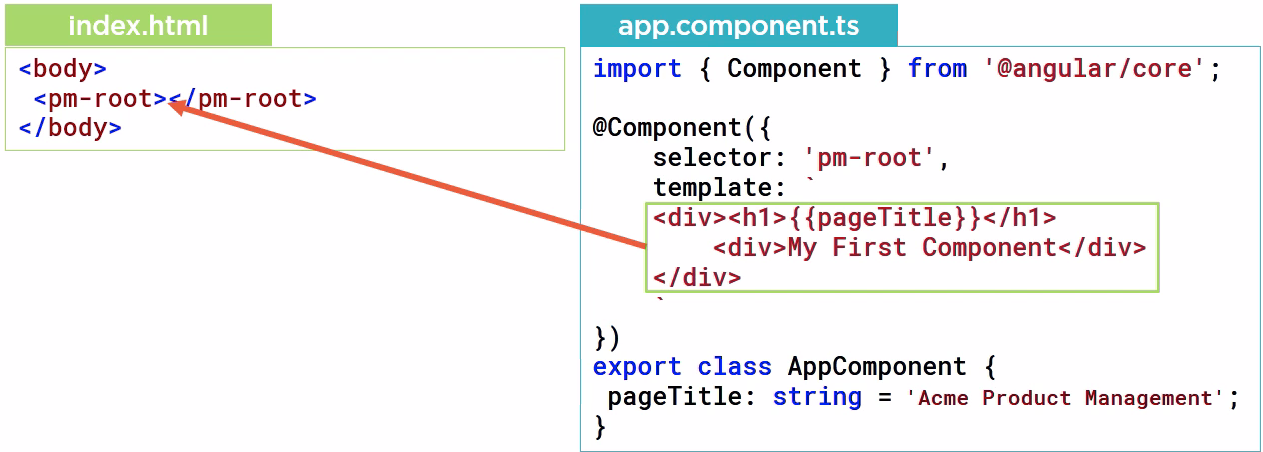


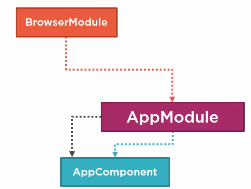
The @Component() decorator function needs to be imported from Angular Core.

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



How does everything work:





AppModule = app.module.ts(This is the main angular module)

AppComponent = app.component.ts defined earlier.

**AppModule – Root Module of angular app.**

Uses @NgModule() decorator function.

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

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

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

@NgModule({

declarations: [AppComponent],

/\*

Necessary for all browser apps

\*/

imports: [BrowserModule],

/\*\*

\* startup component which contains selector

\* to display in html, in this case

\* <pm-root> in AppComponent

\* \*/

bootstrap: [AppComponent]

})

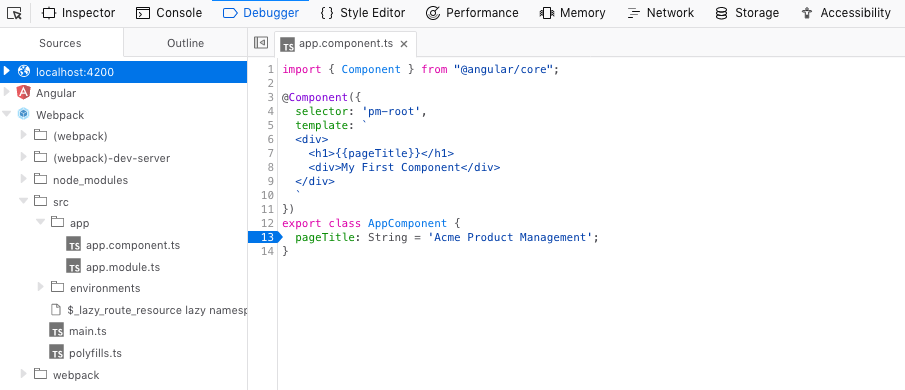
export class AppModule { }

**Debugger:**

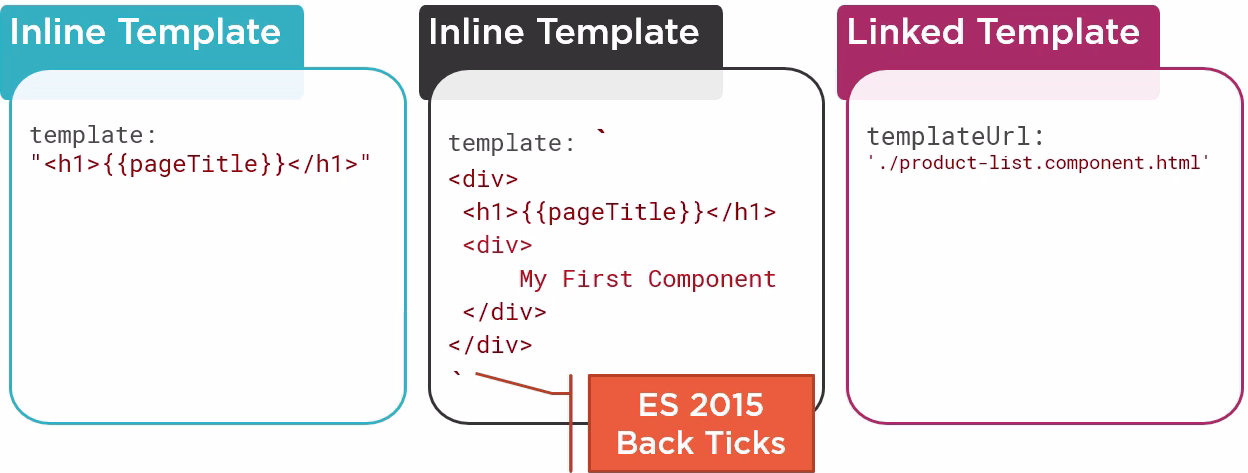
Since Angular CLI uses webpack, debugging can be directly done in browser.

Debugger>Webpack>src>app>codefile(Firefox)

Source>Webpack>src>app>codefile(Firefox)

****

**Many ways to build a template:**

****

Let us install bootstrap and font-awesome for styling

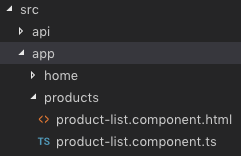
**npm install bootstrap font-awesome**

Then to import their styling properties we import the css in global **src/style.css**

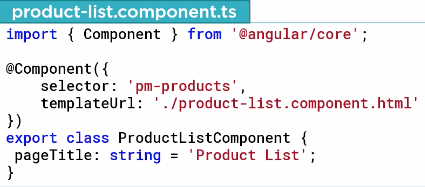
@import "~bootstrap/dist/css/bootstrap.min.css";

@import "~font-awesome/css/font-awesome.min.css";

By convention, each feature has a folder under app folder.



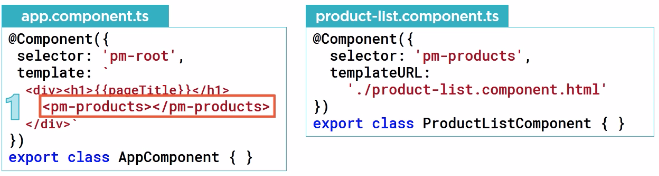
**The new product component.**



**Now we have to add the component as a directive.**

A directive is a custom element which is used to extend html. E.g.: <pm-products>

**Step 1. We add pm-products as html tag in AppComponent**

****

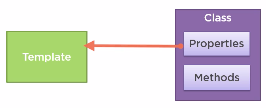
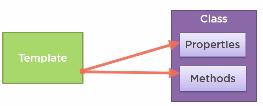
**Step 2. Expose the directive by declaring it in module or importing it if already declared.**

****

**We declare ProductListComponent in AppModule.**

**Binding in Angular JS.**

Binding is used to communicate between a component’s class and its template and involves passing data.

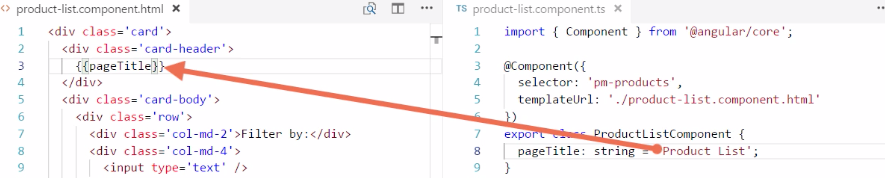
 

1. **Interpolation – one-way binding from class property to template defined by**

**{{}}**

****

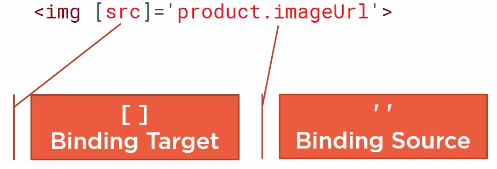
The text between curly braces is called a template expression which is resolved by angular in the context of corresponding component. e.g. Angular will search for a property called pageTitle here in AppComponent because the template containing pageTitle is in AppComponent class.



Different interpolation examples:

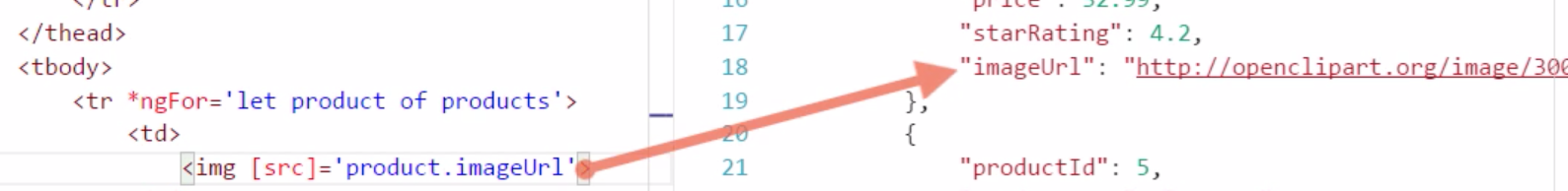


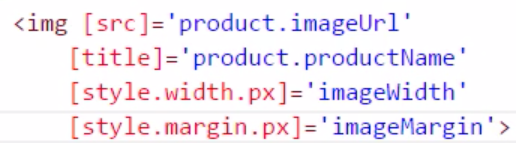
1. **Property Binding**

****

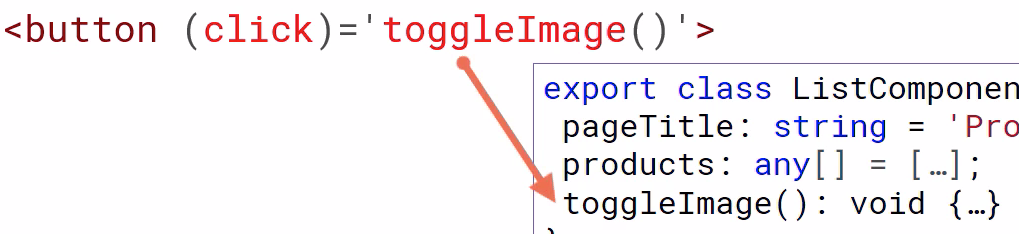
****

The last 2 examples are property binding using interpolation.





1. **Event Binding**



We declare and Boolean variable

showImage: boolean = false;

and a method

toggleImage(): void {

this.showImage = !this.showImage;

}

in ProductListComponent.ts.

Then in product-list.component.html

We define the action:

<th><button (click)='toggleImage()' class="btn btn-primary">

{{showImage ? 'Hide' : 'Show'}} Image</button></th>

And the element affected is manipulated by an \*ngif which evaluates to the Boolean variable defined earlier.

<img

[src] = 'product.imageUrl'

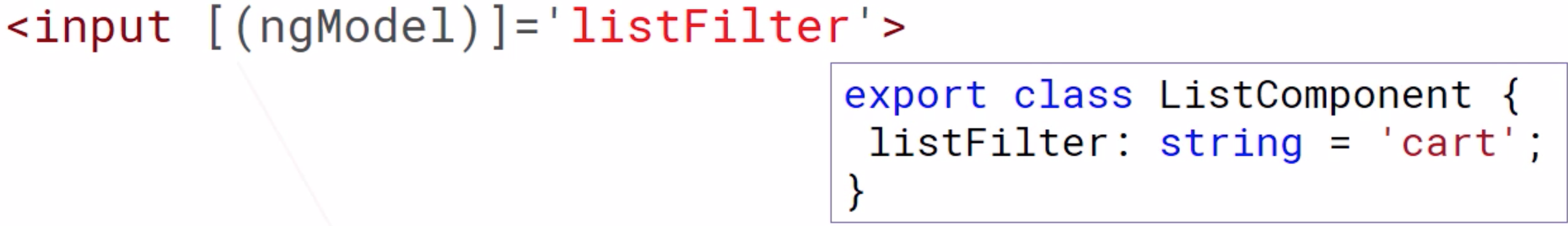
[title] = 'product.productName'

[style.width.px]='imageWidth'

[style.margin.px]='imageMargin'

\*ngIf='showImage'>

1. **Two-way binding**



To use ngModel , we need to import FormsModule in appModule.



In the HTML, we add the ngModel to bind the input in textArea to listFilter variable in component and then display it back in HTML using interpolation {{listFilter}}

Then we add forms module to app module.

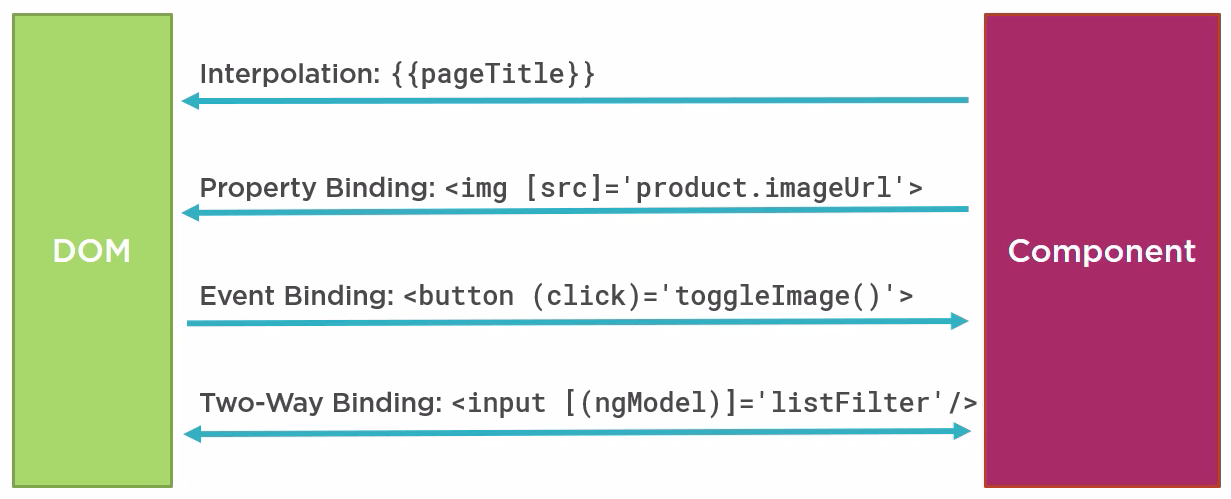
import { FormsModule } from '@angular/forms';

imports: [

BrowserModule,

FormsModule

],



**Angular Built in directives – if and for:**

These are structural directives marked by a \*.

They are equal to a quoted string.

They are exposed by the browser module.

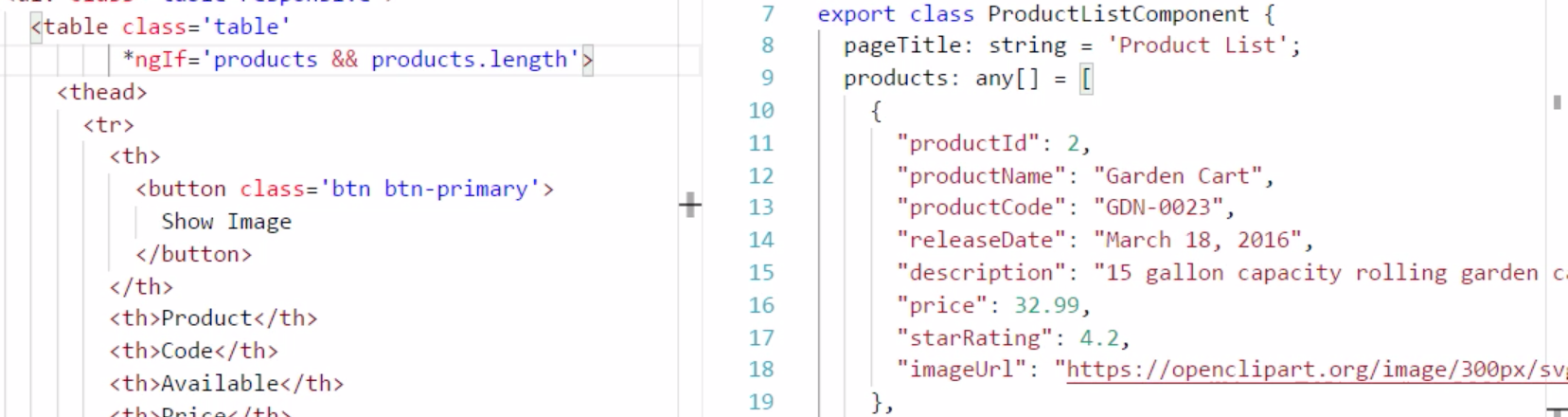
****

<table class='table' \*ngIf='products && products.length'>

**\*ngIf** inserts elements in a dom if the expression evaluates to a true value

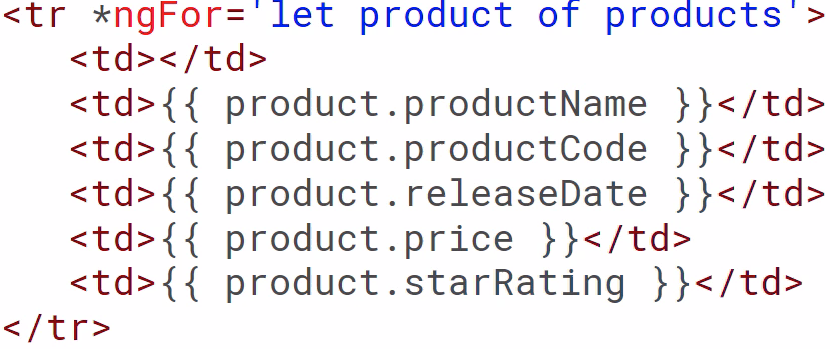
and removes elements from a dom if the expression evaluates to a false value.

So here if the array products in ProductListComponent is not null and has some content, the table will be rendered.



So, now if there is nothing in products, the table will not be rendered.

**\*ngFor** repeats a portion of the dom tree, once for each item in an iterable list.



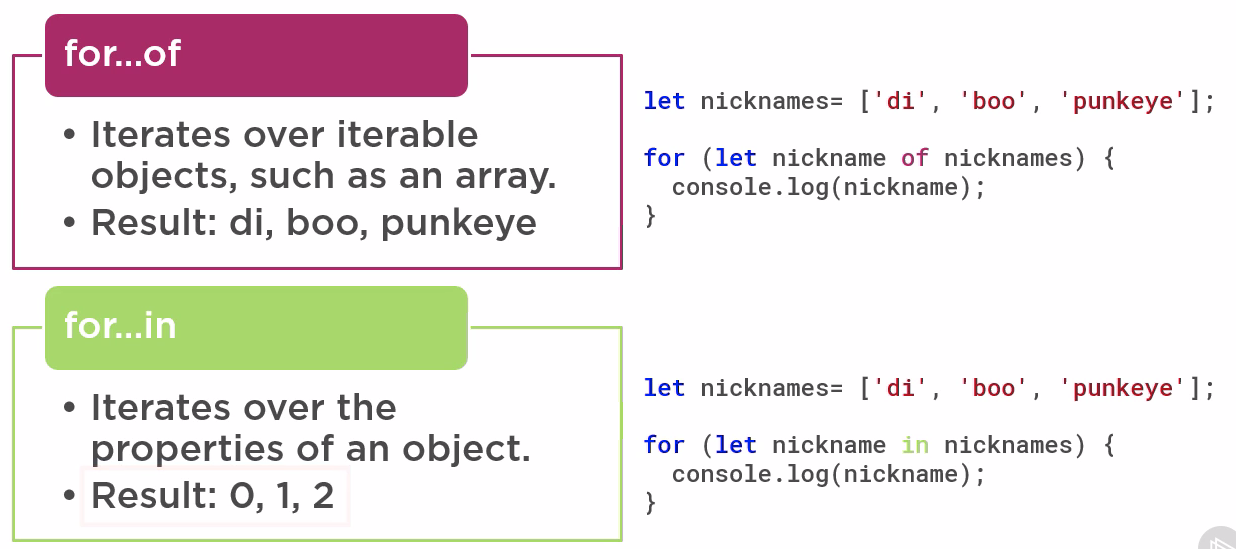
So here, a row will be inserted for each element in products.

The let keyword creates a variable product for each element in products.

Then various attributes of that variable can be accessed

\*ngFor is of 2 types.

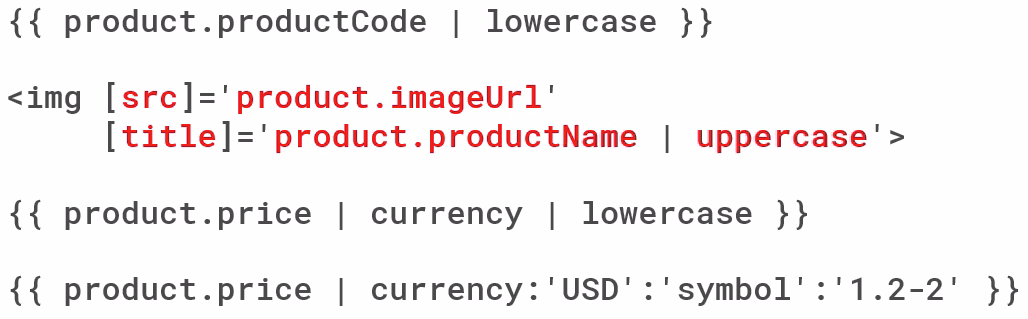
**For .. of shows context and For .. in shows index.**



**Transform Data with pipes:**

Pipes transform bound data before display. There are built in pipes for decimal, currency etc.

Example:



* Pipes can be applied to properties.
* They can be changed together.
* They are also parameterized. Currency:’USD’:’symbol’:’1.2-2’ (The 1st parameter USD tells locale, the 2nd one symbol tells how to display (here $), the 3rd one shows digits – 1.2-2 means minimum 1 digit to the left of decimal, minimum 2 digits to the left of decimal and maximum 2 digits to the left of decimal.

**Custom pipes:**

1. We create a custom pipe using a @Pipe decorator, giving it a name(here convertToSpaces), implementing the PipeTransform interface and implementing the transform()method which described the operation the pipe does.

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

@Pipe({

name: 'convertToSpaces'

})

export class ConvertToSpacesPipe implements PipeTransform{

transform(inputValue: string, characterToReplace: string): string {

return inputValue.replace(characterToReplace,' ');

}

}

2. Use the pipe in html.

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

3. Declaring and importing the pipe in module containing the component, here appModule.

@NgModule({

declarations: [

AppComponent,

ProductListComponent,

ConvertToSpacesPipe

],

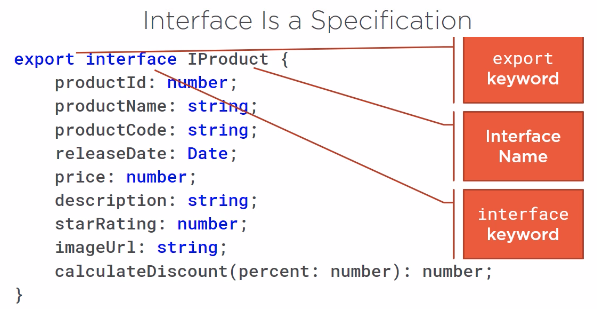
**Angular Interface**

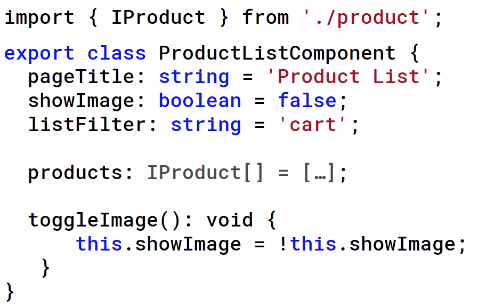
The interface is a specification identifying methods and properties.

A class implementing the interface has to support this spec by implementing the methods.

The interface is used as a data type.

It is development time only, it is not trans piled to JavaScript.





Here the IProduct interface is imported and then used as

Products : Iproduct[] = […]

A class can also implement the interface and then be used.

e.g:

import { Product } from "./product";

export class ProductImplementation implements Product{

constructor(

public productId: number,

public productName: string,

public productCode: string,

public releaseDate: string,

public description: string,

public price: number,

public starRating: number,

public imageUrl: string){

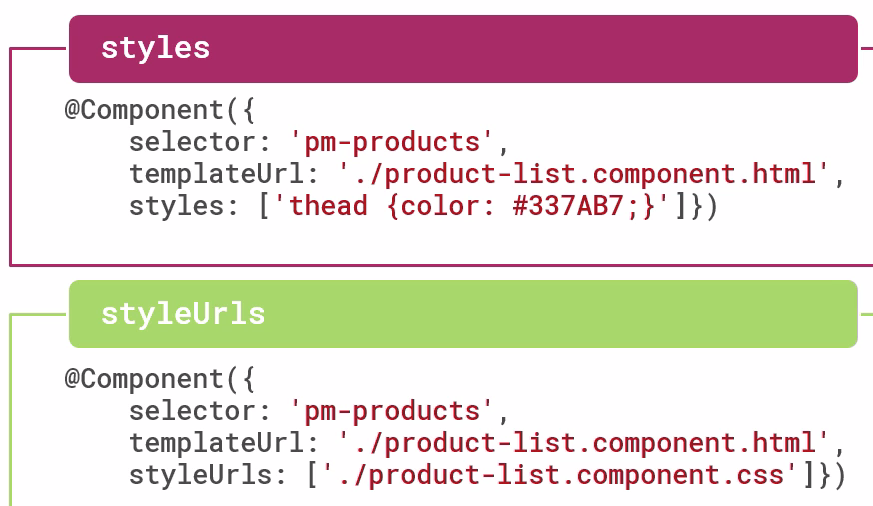
}

}

products: ProductImplementation[] = [..]

**Encapsulating Component styles:**

The styles for a component can be encapsulated using the styles or styleUrls attribute of the @Component decorator.



We put the styleUrls property of @Component in productListComponent and add a stylesheet

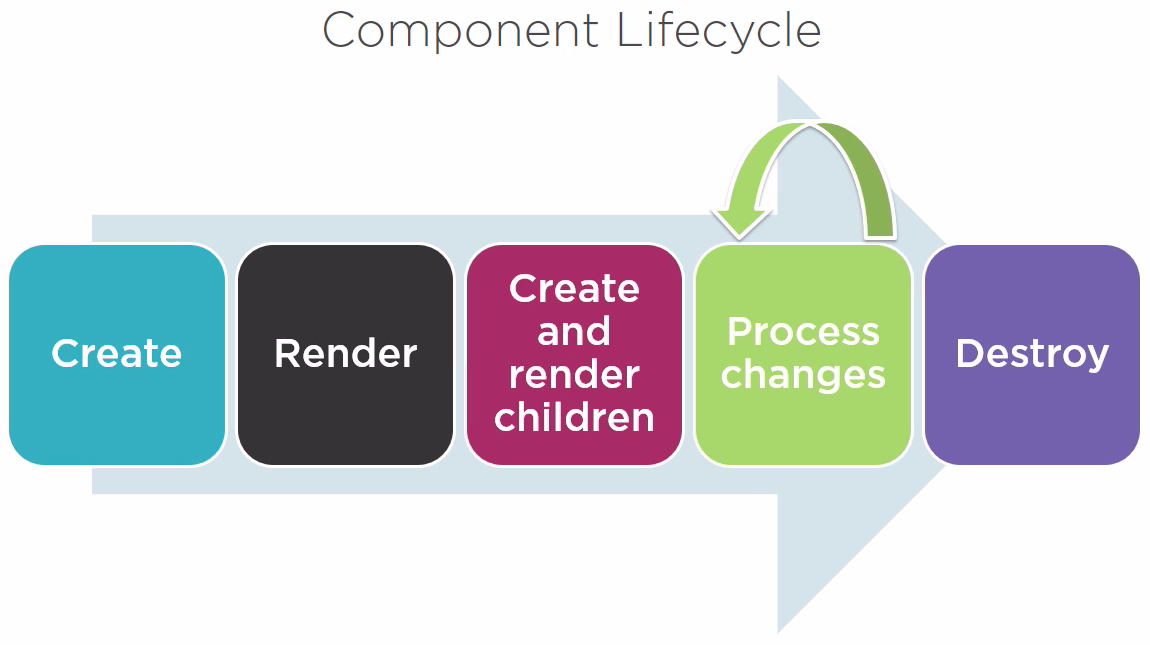
Which has the following code to make tableheader blue.

thead{

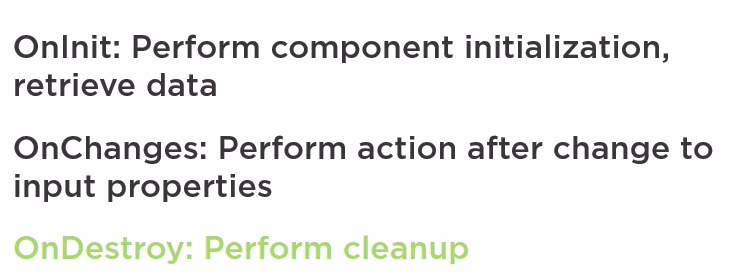
color: #337AB7

}

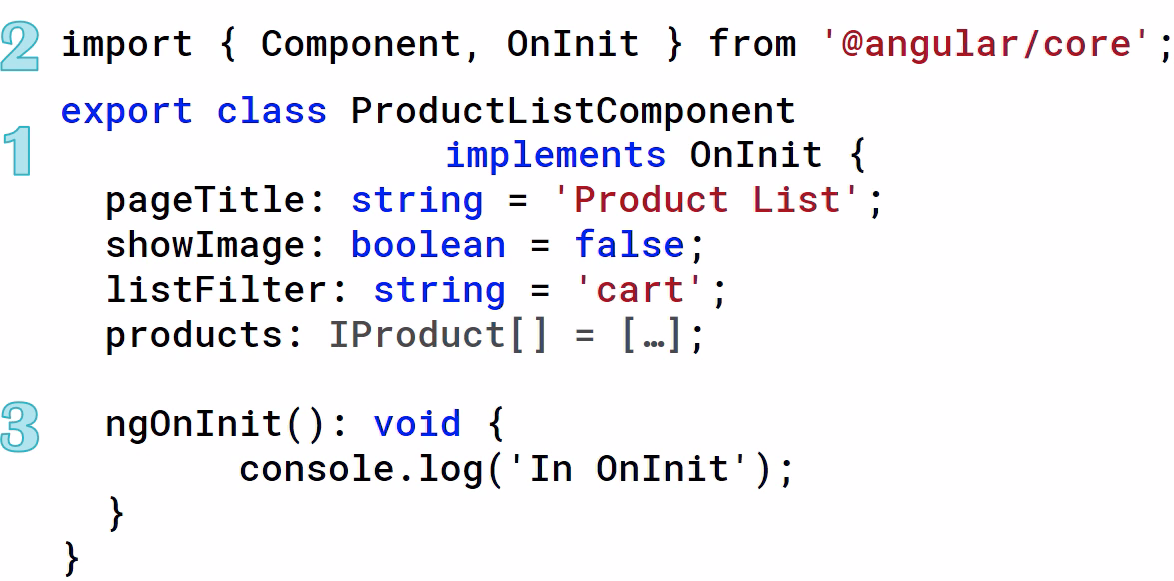
**Angular Component Lifecycle hooks:**

****

We can manipulate the angular component lifecycle using lifecycle hooks.



How to use them:



* Step 1 is implementing the hook (optional, we can skip this and still ngOninit works)
* Step 2 is importing from angular/core
* Step 3 is implementing the hook method.

**Filtering values based on input text box:**

1. Generate getter and setter of \_listFilter and create a new filteredProducts[]

private \_listFilter: string = 'cart';

public get listFilter(): string {

        return this.\_listFilter;

    }

public set listFilter(value: string ) {

this.\_listFilter = value;

this.filteredProducts = value ? this.performFilter(value) : this.products;

}

filteredProducts: ProductImplementation[];

In the setter method call performFilter method to filter products[] appropriately

And assign the filtered array to filteredProducts[]

1. Write performFilter() method to filter array

performFilter(filterBy: string ): ProductImplementation[]{

filterBy = filterBy.toLocaleLowerCase();

return this.products.filter((product:ProductImplementation) =>

product.productName.toLocaleLowerCase().indexOf(filterBy)!== -1);}

1. Assign default values using class constructor

constructor(){

this.\_listFilter = 'cart';

this.filteredProducts = this.products;

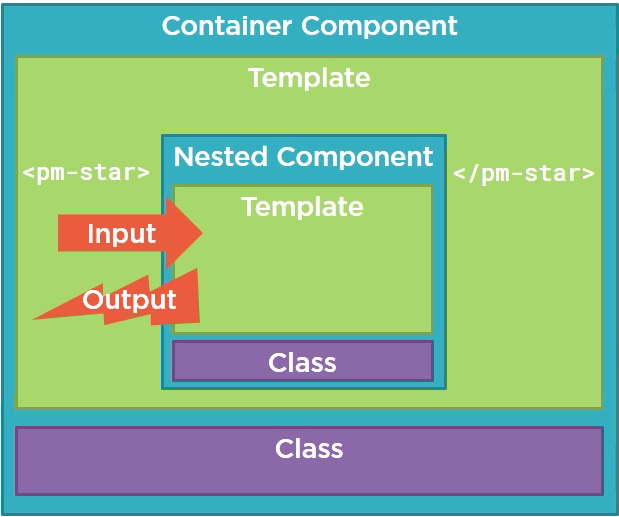
}

1. Change assignment in html

<tr \*ngFor='let product of filteredProducts'>

<table class='table' \*ngIf='filteredProducts && filteredProducts.length'>

**ANGULAR NESTED COMPONENTS**



1. **We create the nested component along with its stylesheets and html.**

**The component**

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

@Component({

selector: 'pm-star',

templateUrl: './star.component.html',

styleUrls:['./star.component.css']

})

export class StarComponent implements OnChanges{

starWidth: number;

ngOnChanges(): void {

this.starWidth= this.rating \* 75 /5;

}

}

**The HTML and CSS:**

.crop {

overflow: hidden;

}

div {

cursor: pointer;

}

<div class="crop"

[style.width.px]="starWidth">

<div style="width: 75px">

<span class="fa fa-star"></span>

<span class="fa fa-star"></span>

<span class="fa fa-star"></span>

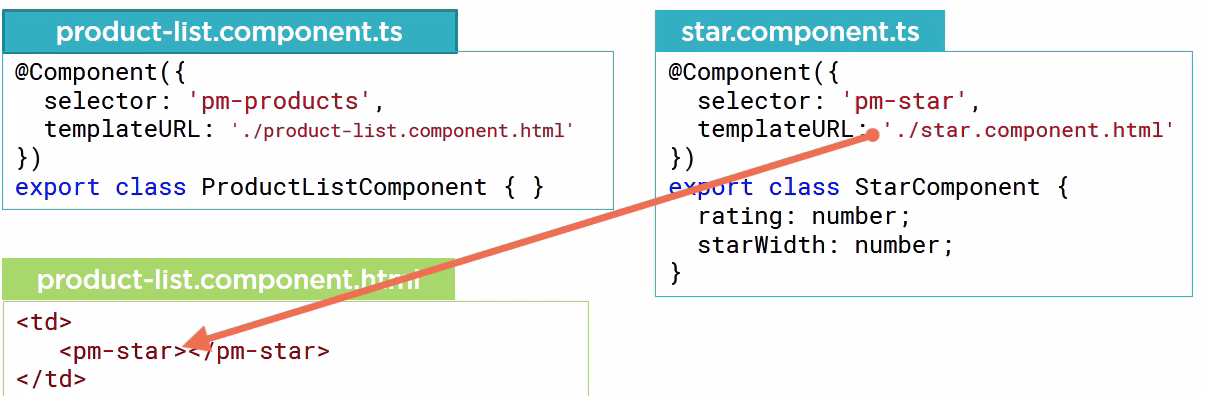
<span class="fa fa-star"></span>

<span class="fa fa-star"></span>

</div>

</div>

1. **We include it in another component as a directive.**



Include the component in the app module.

@NgModule({

declarations: [

AppComponent,

ProductListComponent,

ConvertToSpacesPipe,

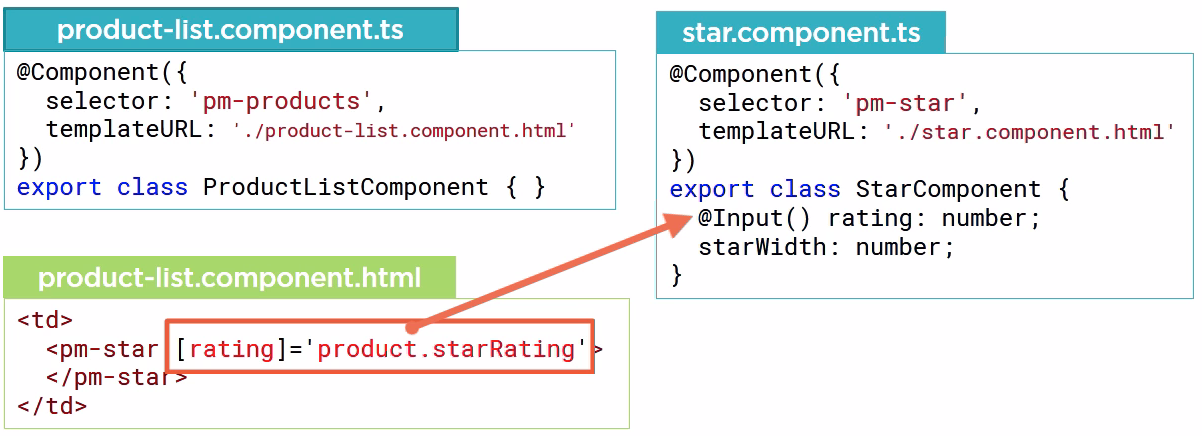
StarComponent

],

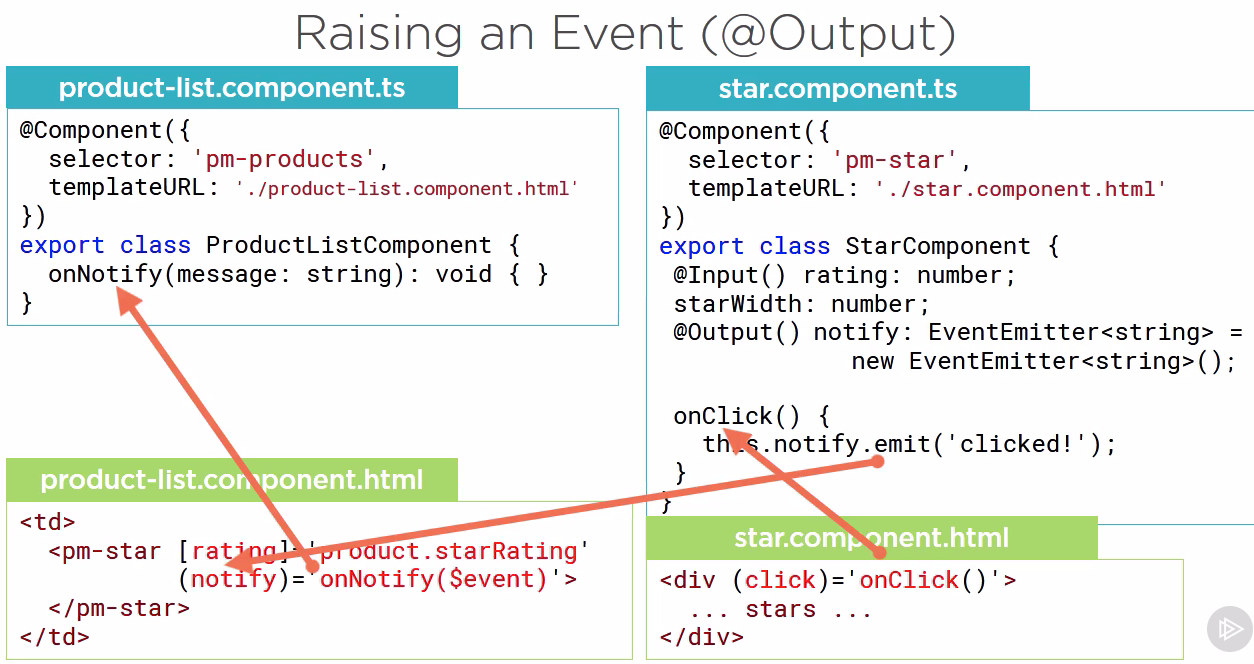
1. **We create input properties so that we can send data from container component to nested component, in this case rating so that onchanges() method can listen to changes and calculate starwidth according to rating passed as input.**

@Input() Decorator marks rating in starComponent as a input property so that the container component productListComponent can pass values to the rating property

to starComponent.



1. We pass data(payload) back to container component using events by means of an @output decorator.



1. We need to respond to users click in starcomponent html.

<div class="crop"

[style.width.px]="starWidth"

[title]="rating"

(click) = 'onClick()'>

<div style="width: 75px">

<span class="fa fa-star"></span>

<span class="fa fa-star"></span>

<span class="fa fa-star"></span>

<span class="fa fa-star"></span>

<span class="fa fa-star"></span>

</div>

</div>

1. Define event and onClick() in starcomponent class.

import { Component, OnChanges, Input, Output, EventEmitter } from '@angular/core';

@Component({

selector: 'pm-star',

templateUrl: './star.component.html',

styleUrls:['./star.component.css']

})

export class StarComponent implements OnChanges{

starWidth: number;

@Input() rating: number;

@Output() ratingClicked: EventEmitter<string> = new EventEmitter<string>();

ngOnChanges(): void {

this.starWidth= this.rating \* 75 /5;

}

onClick(){

console.log(`The rating ${this.rating} was clicked`);

this.ratingClicked.emit(`The rating ${this.rating} was clicked`);

}

}

Here ratingClicked is the Event of type EventEmitter. onClick() of start component, this event emits a string, **The rating ${this.rating} was clicked** to the container component.

c)The container component receives the data by using two-way event binding.

<td>

<pm-star [rating] = 'product.starRating'

(ratingClicked) = 'onRatingClicked($event)'>

</pm-star>

</td>

d)The container Component then needs to implement method onRatingClicked() to implement operations to be done with the data passed in event. It displays the data in page title here.

onRatingClicked(message:String):void{

this.pageTitle = 'Product List: ' + message;

console.log(message);

}