ANGULAR 6

Lesson 05

PIPES, SERVICES & DEPENDENCY INJECTION

Objectives

Capgemini

Pipes, Services & Dependency Injection

- Parametrized Pipes
- Chaining Multiple Pipes
- Creating a Custom Pipe
- Creating a Filter Pipe
- Pure and Impure Pipes (or: How to "fix" the Filter Pipe)
- Understanding the "async" Pipe
- Services
- Dependency Injections
- Creating Data Service
- Understanding Hierarchical Injector
- Services for Cross Component Communication
- Injection Tokens

PIPES

Pipe

A pipe takes in data as input and transforms it to a desired output.

Pipes transform bound properties before they are displayed

Angular pipes, a way to write display-value transformations that we can declare in your HTML.

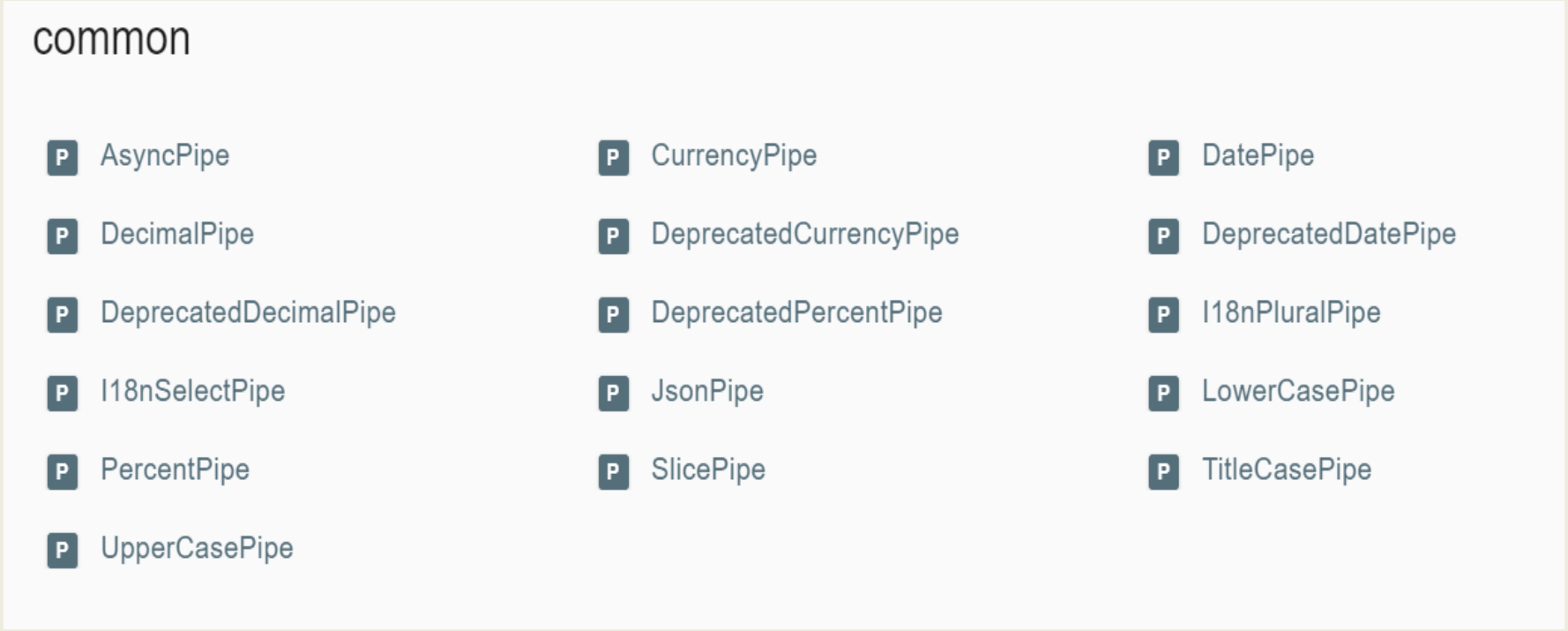
To pass an argument to a pipe in the HTML form, pass it with a colon after the pipe (for multiple arguments, simply append a colon after each argument)

Angular gives us several built-in pipe like lowercase, date, number, decimal, percent, currency, json, slice etc

Angular provides a way to create custom pipes as well.

Build in Pipes

https://angular.io/api?type=pipe-- We Can see all Build in Pipes



Chaining Pipes & Parameterizing a pipe

- We can chain pipes together in potentially useful combinations.
- A pipe may accept any number of optional parameters to fine-tune its output.
- We add parameters to a pipe by following the pipe name with a colon (:) and then the parameter value (e.g., currency:'EUR').
- If our pipe accepts multiple parameters, we separate the values with colons (e.g. slice:1:5).

```
<tr *ngFor="let emp of employess" >
<td>{{emp.empId}}</td>
<td>{{emp.empName |uppercase | slice:1:3}}</td> <!-- Channing pipes & passing parameter
-->
<td>{{emp.empSal | currency:'USD':true}}</td> <!-- Currency pipes -->
<td>{{emp.empDep}}</td>
<td>{{emp.empjoiningdate | date:'fullDate'|uppercase}}</td> <!-- Dates pipes channing pipes
-->
```

Creating a Custom Pipe

- A pipe is a class decorated with pipe metadata.
- The pipe class implements the PipeTransform interface's transform method that accepts an input value followed by optional parameters and returns the transformed value.
- There will be one additional argument to the transform method for each parameter passed to the pipe. Your pipe has one such parameter: the exponent.

Creating a Custom Pipe (Contd...)

Capgemini

- To tell Angular that this is a pipe, you apply the @Pipe decorator, which you import from the core Angular library.
- The @Pipe decorator allows you to define the pipe name that you'll use within template expressions. It must be a valid JavaScript identifier.

```
import {PipeTransform,Pipe} from 'angular2/core';

@Pipe({  name : 'customPipe'})

export class ExponentialStengthPipe implements PipeTransform{
        transform(value:number,args:string[]):any {
            return  Math.pow(value,parseInt(args[0] || ‘1’, 10));
             }
}
```

Demo

- Demo Build In Pipes
- Demo Custom Pipe

Creating a Filter Pipe

Pure and Impure Pipes
How to "fix" the Filter Pipe

Understanding the "async" Pipe

SERVICES & DEPENDENCY INJECTION

Dependency Injection

- Problem without DI

```
class Engine{
    constructor(newparameter){}
}
class Tires{
    constructor(){}
}
```

```
class Car{
    engine;
    tires;
    constructor()
    {
        this.engine = new Engine();
        this.tires = new Tires();
    }
}
```

Dependency Injection (Contd...)

- Dependency injection is an important application design pattern
- Angular has its own dependency injection framework
- DI allows to inject dependencies in different components across applications, without needing to know, how those dependencies are created, or what dependencies they need themselves.
- DI can also be considered as framework which helps us out in maintaining assembling dependencie for bigger applications.

```
class Car{
    engine;
    tires;
    constructor()
    {
        this.engine = new Engine();
        this.tires = new Tires();
    }
}
```

```
class Car{
    engine;
    tires;
    constructor(engine, tires)
    {
        this.engine = engine;
        this.tires = tires;
    }
}
```

Dependency Injection (Contd...)

Example 1 : DI

```
public description = 'DI';

constructor(public engine: Engine, public tires: Tires) { }
```

Example 2 : DI

```
class Engine2 {

  constructor(public cylinders: number) { }

}

let bigCylinders = 12;

let car = new Car(new Engine2(bigCylinders), new Tires());
```

Services

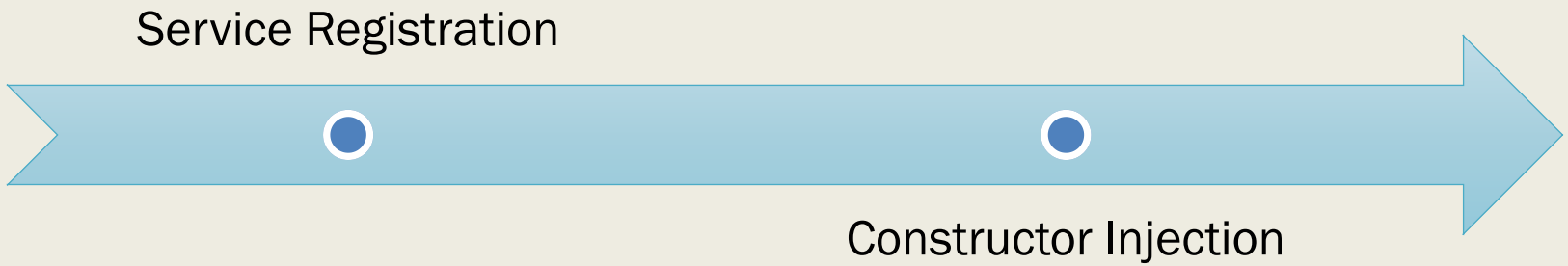
- Services provided architectural way to encapsulate business logic in a reusable fashion.
- Services allow to keep logic out of your components, directives and pipe classes
- Services can be injected in the application using Angular's dependency injection (DI).
- Angular has In-built service classes like Http, FormBuilder and Router which contains logic for do specific things that are non component specific.
- Custom Services are most often used to create Data Services.

Working with Services in Angular 6

- Component can work with service class using two ways
  - *Creating an instance of the service class*
    - Instances are local to the component, so data or other resources cannot be shared
    - Difficult to test the service
  - *Registering the service with angular using angular Injector*
    - Angular injector maintains a container of created service instances
    - The injector creates and manages the single instance or singleton of each registered service.
    - Angular injector provides or injects the service class instance when the component class is instantiated. This process is called dependency injection
    - Angular manages the single instance any data or logic in that instance is shared by all of the classes that use it.
    - This technique is the recommended way to use services because it provides better management of service instances it allow sharing of data and other resources and it's easier to mock the services for testing purposes

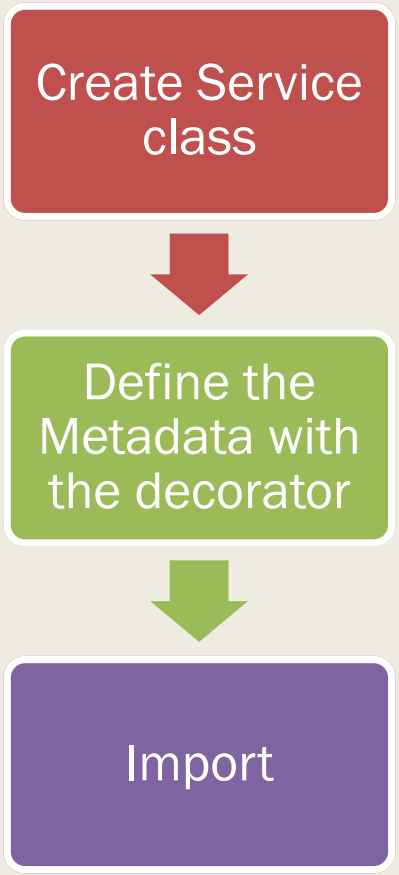
Working with Services in Angular 6

- Angular has dependency injection support baked into the framework which allows to creat component directives in modular fashion.
- DI creates instances of objects and inject them into places where they are needed in a two ste process.



Building a Service

- Steps to build a service is similar to build components and a custom pipe.
- It is recommended that every service class use the injectable decorator for clarity and consistency.
- ***@Injectable*** is a decorator, that informs Angular 2 that the service has some dependencies itself. Basically services in Angular 2 are simple classes with the decorator @Injectable on top of the class, that provides a method to return some items.



Steps to Create services

Create the Service File

Import the Injectable Member

- *import { Injectable } from '@angular/core';*

Add the Injectable Decorator

- *@Injectable()*

Export the Services Class

Import the Services to component

Add it as a Provider

- *providers: [ExampleService]*

Include it through depedency injection

- *constructor(private _exampleService: ExampleService) {}*

Capgemini

Understanding Hierarchical Injector

Services for Cross Component Communication

Injection Tokens