HEXAWARE

Angular –Services and Dependency Injection

Session Plan

- Introduction to Services
- Building the Service
- Registering the Service
- Dependency Injection
- Injecting the Service
- Life cycle hooks

Introduction to Services

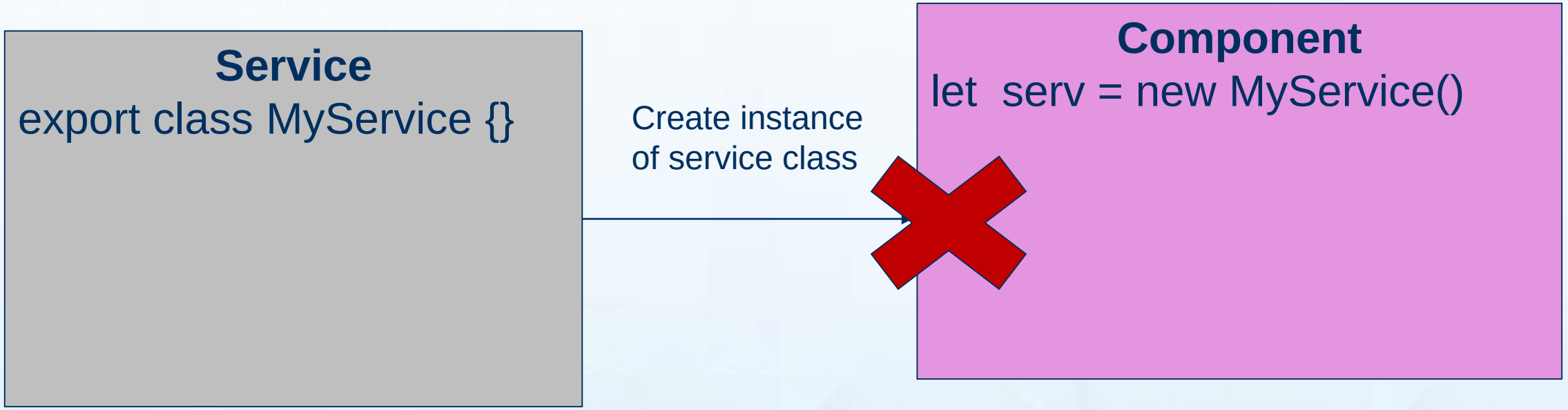
- The data or logic are not associated with a specific view or that we want to share across components. We build services.
- Services is a class with a focused purpose.

-Used for features that

- are independent from any particular component, to share data or logic.
- provide share data or logic across components
- encapsulate external interactions such as data access.

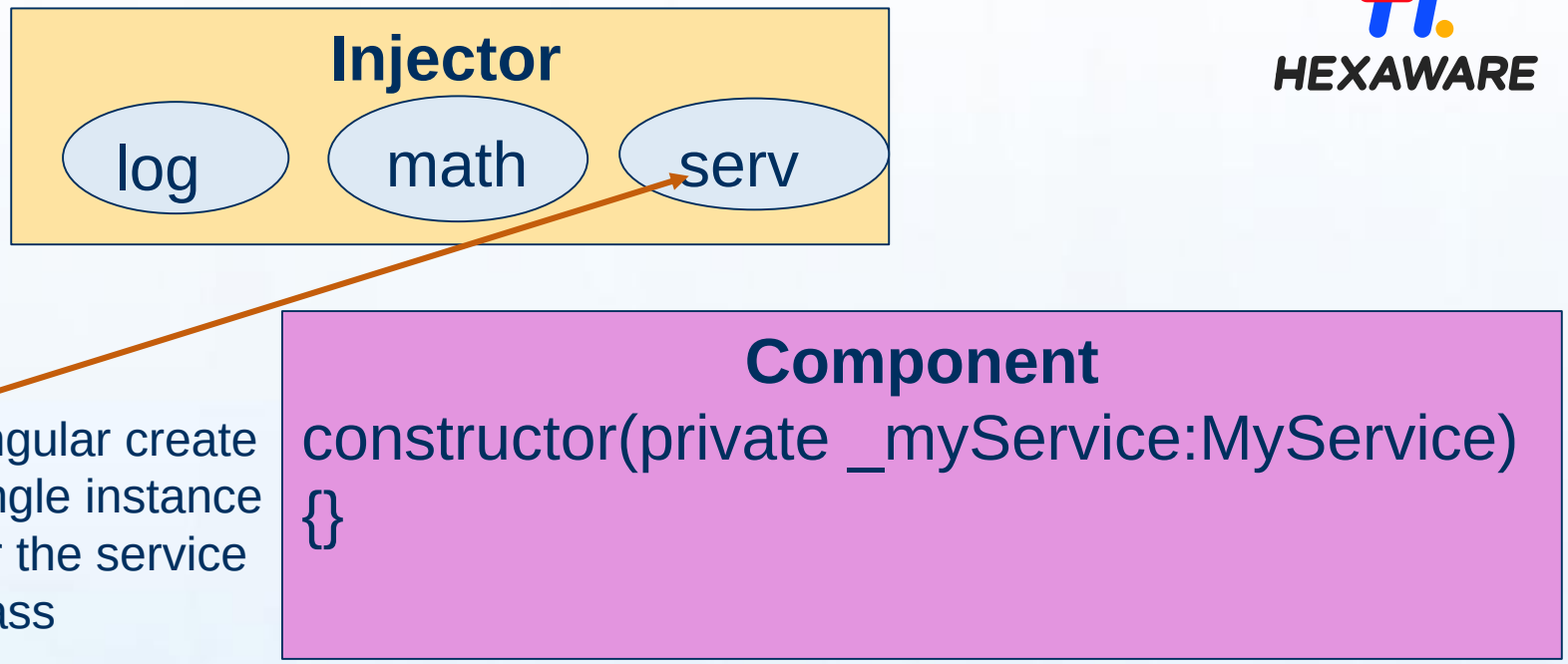
By shifting these responsibility from the component to a service, the code easy to test, debug and reuse.

How Does it Work?



- The instance is local to the component.
- We can’t share data or other resources.
- It will be more difficult to mock the service for testing
- Alternatively register the service with Angular.

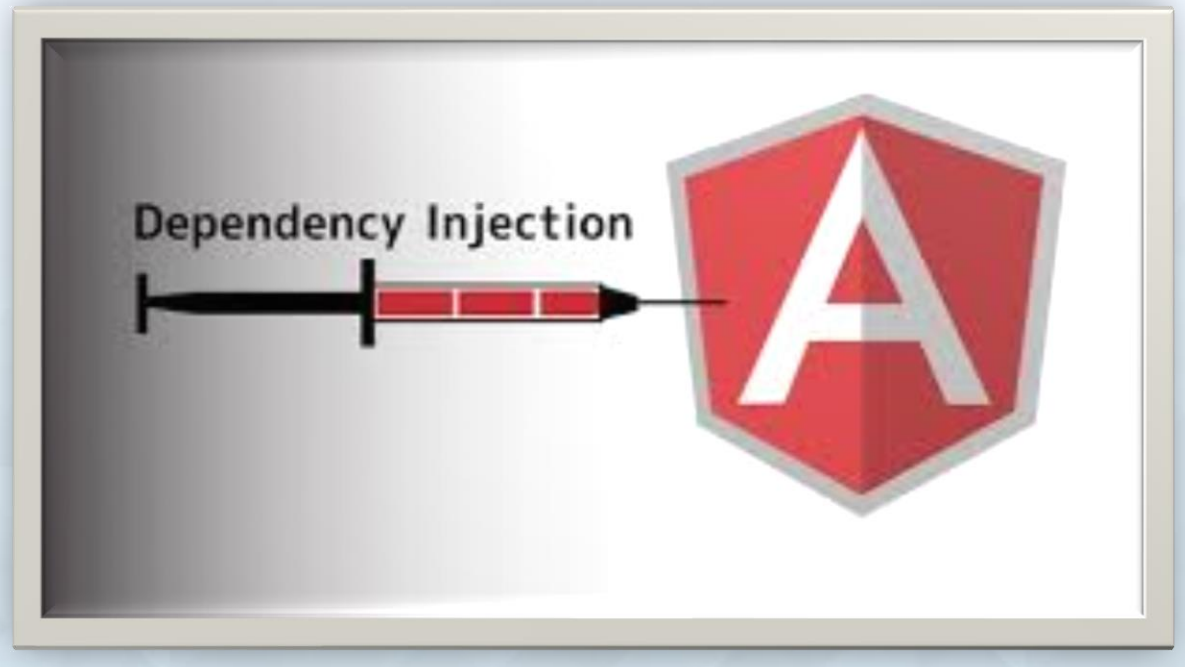
How Does it Work?



- Angular provides build in Injector.
- We register our services with the Angular Injector. Which maintains a container of created services instances.
- Injector creates & manage single instance or **singleton** for each registered service.
- The Angular Injector injects the service class instance when the component class is instantiated. This process is called **Dependency Injection**.

Dependency Injection

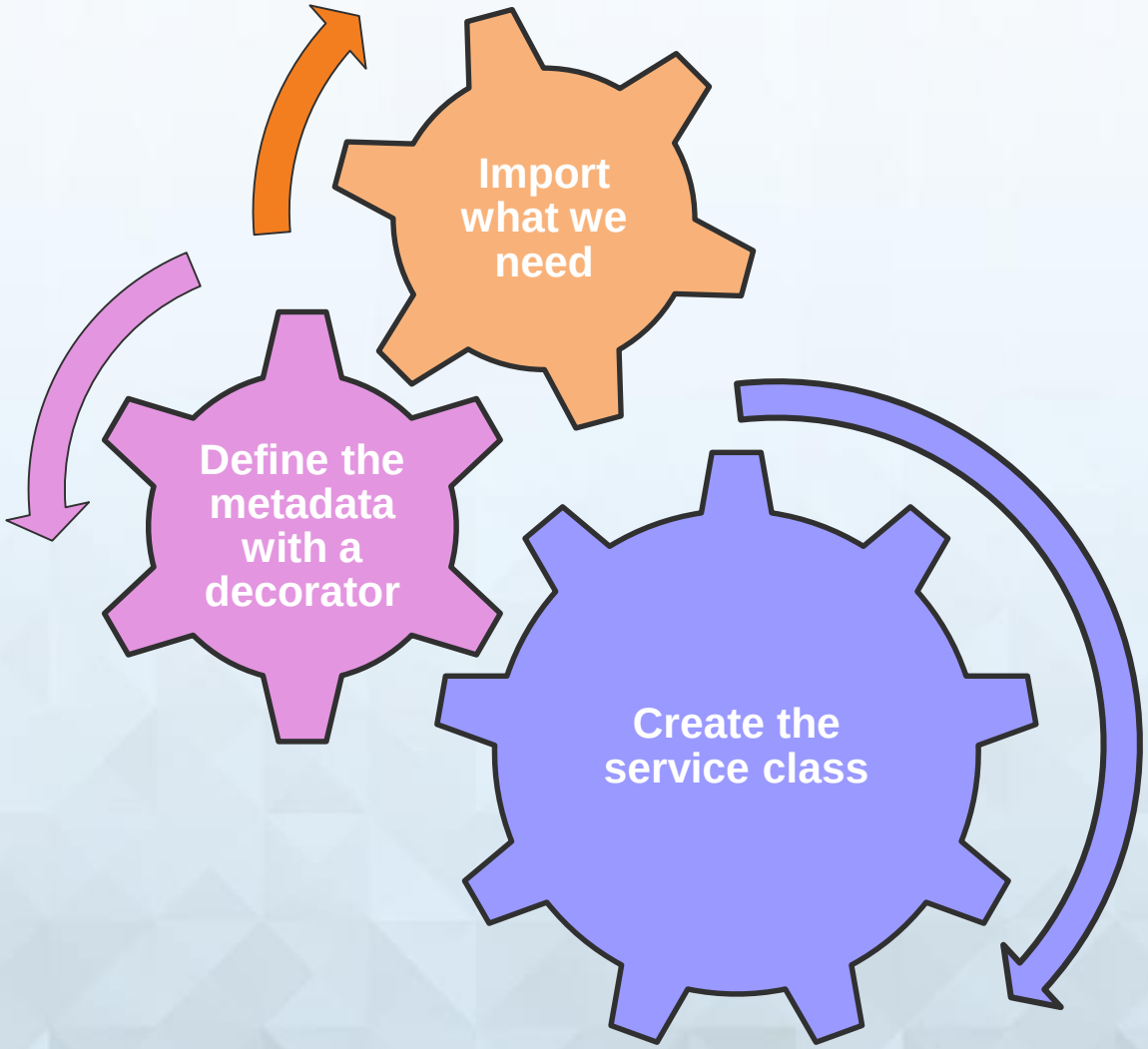
- A coding pattern in which a class receives the instances of object it needs (called dependencies) from an external source rather than creating them itself.
- In Angular this external source is the **Angular Injector**.



Steps for creating the Service

- Building a Service
- Registering the Service
- Injecting the Service

Building a Service



Building a Service (Cont..)

product.service.ts

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

@Injectable()
export class ProductService
{
    getProducts():Iproduct[]{
     return;
     }
}
```

Registering the Service

- **Register a provider**
  - Code that can create or return a service
  - Typically the service class itself

Define in component OR Angular module metadata.

Registered in component:

Injectable to component and its children

Registered in Angular module:

Injectable everywhere in the application.

Registering the Service (Cont..)

```
…
import {ProductService} from './products/product.service';
@Component({
        selector: 'app-component',
        template: './app.component.html',
        styleUrls: ['./app.component.css'],
        providers: [ProductService]})
export class AppComponent {
title = 'app';}
```

Injecting the Service

- Dependency Injection in TypeScript:
  - Perform dependency injection in Constructor.
  - Every class has a constructor that is executed when an instance of the class is created.
  - If there is no explicit constructor defined for the class, an implicit constructor is used.
  - But if we want to inject the dependencies such as an instance of a service, We need an explicit constructor.
  - In TypeScript a constructor is defined with a constructor function.

Injecting the Service

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

@Component({
selector: 'app-sample',
templateUrl: './product-list.component.html',
})
export class ProductListComponent {

constructor(private _sampleService: SampleService)
 { }
 }
```

CREATING SERVICE Example

Creating Service

Steps to create Service

1. Create an Interface Iemployee.ts
2. Create a Service class Employee.service.ts
3. Create a Component class Employee.component.ts
4. Create a module app.module.ts

Creating Service cont…

Step 1:

Create an interface **IEmployee.ts**

Interface:

- TypeScript allows to define complex type definitions in the form of interfaces.
- This complex type such as an object contains other properties.

```
export interface IEmployee{
    eid:number;
    ename:string;
    salary:number;
}
```


Creating Service

Step 2.A:

Create a separate class which has the injectable decorator. The injectable decorator allows the functionality of this class to be injected and used in any Angular JS module.

```
@Injectable()
export class EmployeeService{
}
```


Creating Service cont…

Step 2.B:

Create a service file named **Employee.service.ts**

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

@Injectable()
export class EmployeeService {
    getEmp():IEmployee[] {
    return  [
        {eid:31411,ename:"Amy",salary:80000},
        {eid:21222,ename:"Andrew",salary:48000},
        {eid:32212,ename:"vishwa",salary:90000}
    ];
}
}
```

- The Injectable decorator is imported from the angular/core module.
- A class called EmployeeService is created and decorated with the Injectable decorator.
- create a simple function called getEmp(), which returns a array of employee details.

Creating Service cont…

Step 3.A:

Create a file **Employee.component.ts**

```
import { Component } from '@angular/core';
import { EmployeeService } from './Employee.service';
import { IEmployee } from './Employee';
```

Import Employee.service Service module in the Employee.Component module

```
@Component({
  selector: 'Employee',
  templateUrl: 'app/Employee/Employee.component.html'
})
```

Register the Employee.Component.html in the @Component decorator

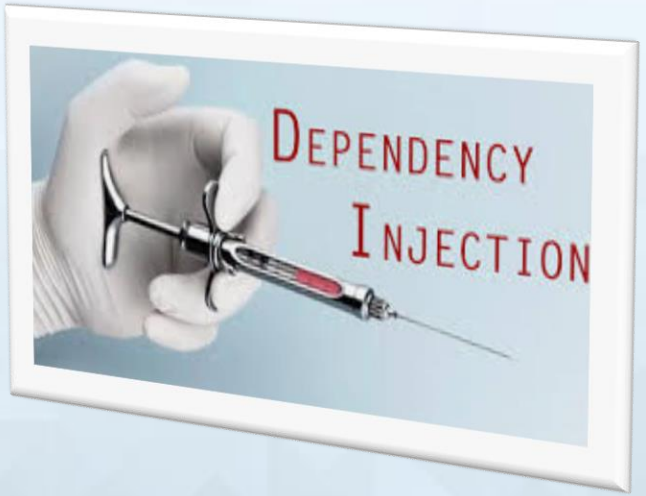
Creating Service cont…

Step 3.B:

Include it through dependency injection

- In the constructor arguments of the component class, we include it through dependency injection:

```
constructor(private _employeeService: EmployeeService) { }
```



Creating Service cont…

Step 3.C:

Using Service

- The service's methods and properties can be referenced using the private _employeeService

```
ngOnInit() {
 this.employee= this._employeeService.getEmp();
}
```


Creating Service cont…

Step 3.D:

```
export class EmployeeComponent{

employee:IEmployee[];
constructor(private _employeeService:EmployeeService ){}
ngOnInit(){
  this.employee=this._employeeService.getEmp();
  console.log(this.employee);
}
}
```

- In the constructor, define a variable called _employeeService of the type EmployeeService so that it can be called anywhere in the Employee.Component module.
- In the ngOnInit lifecyclehook, we called the getEmp() function of the service and assign the output to the value property of the Employee.Component class.

Creating Service cont…

Step 4:

Including the Service in **app.module.ts**

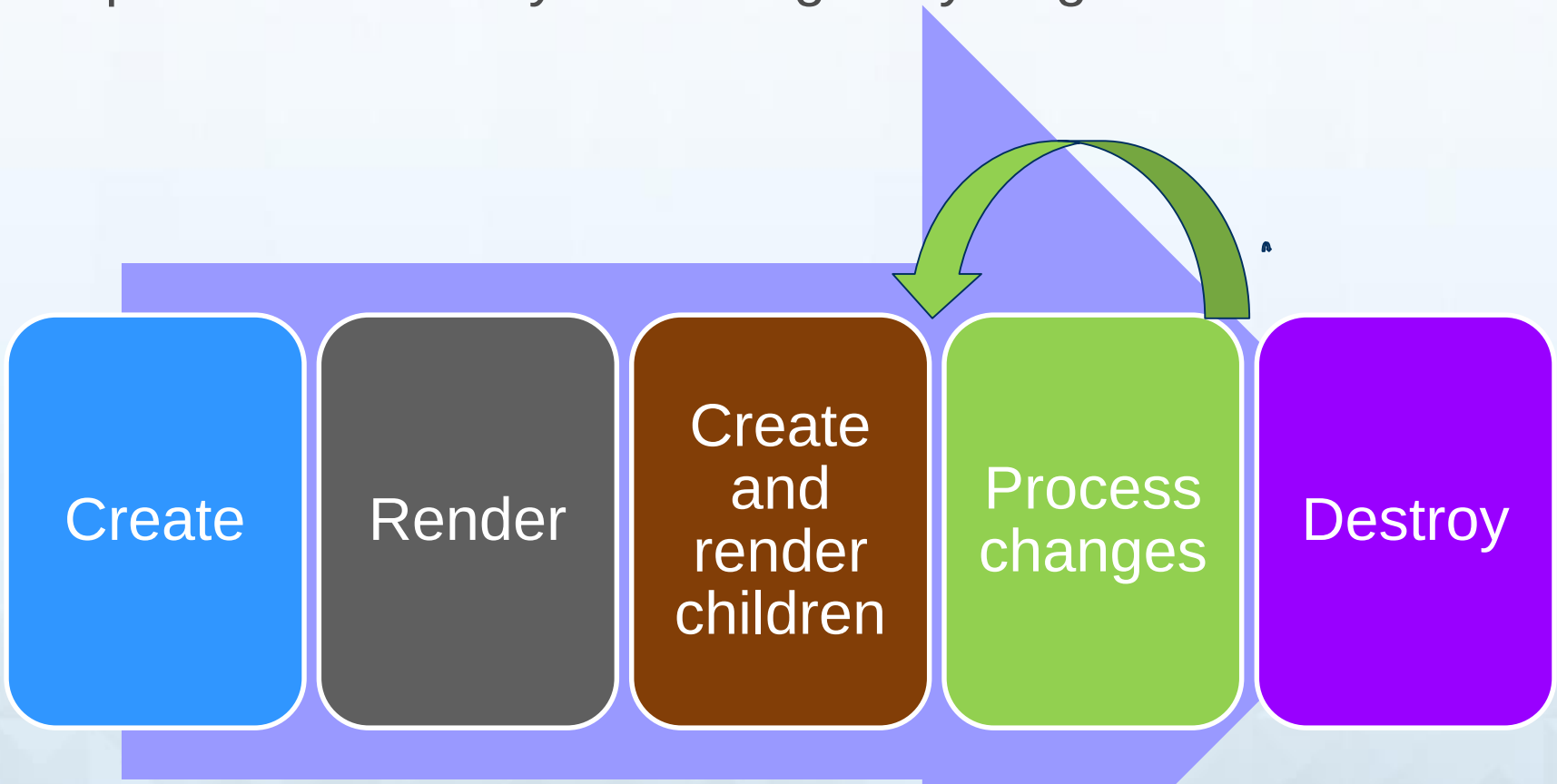
- The difference when including a service in the app.module.ts from including it in a specific component is that, declaring the service in the providers property of the app.module.ts @NgModule metadata, as opposed to the @Component's meta data

```
import { EmployeeService } from './Employee/Employee.service';

@NgModule({
     providers: [EmployeeService]
})
```


Component Life cycle:

- Component has life cycle managed by Angular.



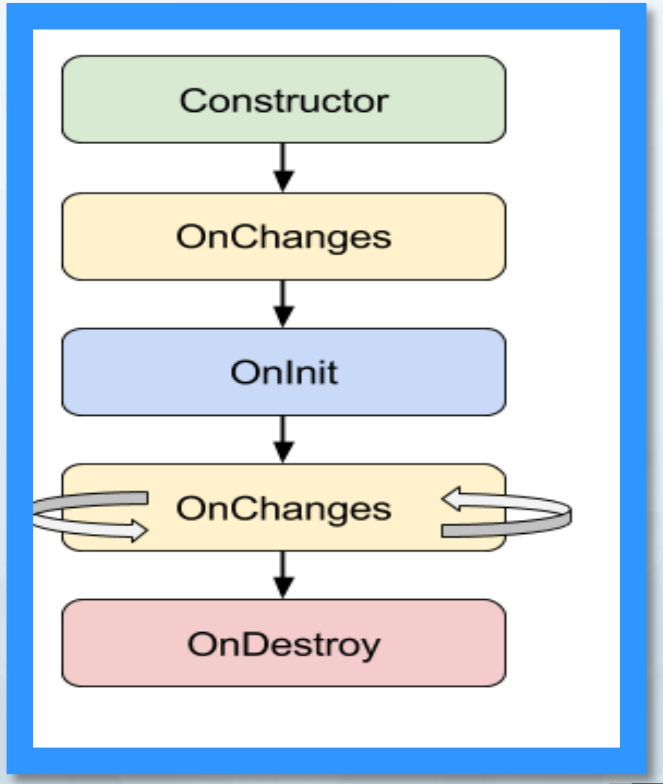
Component Life cycle hooks:

OnInit:

- To Perform Component Initialization, retrieve data.
- Implement this interface to execute custom initialization logic after the directive's data-bound properties have been initialized.
- It is invoked only once when the directive is instantiated.

OnChanges: Perform action after change to input properties.

onDestroy : Perform any clean up.



Component Life cycle hooks:

- Use of ngOnInit() for two main reasons:
  - To perform complex initializations shortly after construction.
  - To set up the component after Angular sets the input properties

HEXAWARE

Innovative Services

Passionate Employees

Delighted Customers

Thank you

www.hexaware.com