Angular Training

(Intermediate to Advanced)

Narasimha

Sr. Corporate Trainer, Mentor

tnrao.trainer@gmail.com

Schedule for React JS Training

Day1 : Custom Pipes in Angular

Day2 : Parent-Child Communication

Day3 : Custom Directives in Angular

Day4 : Working with Reactive Forms

Day5 : Dependency Injection and Services in Angular

Day6 : RxJS Library – Observables

Day7 : Http Client – Server calls in Angular

Day8 : Routing and Security in Angular

Day5
Dependency Injection and Angular Services

Narasimha
Sr. Corporate Trainer, Mentor
tnrao.trainer@gmail.com

Index – Day5

1. What is Dependency Injection
2. How does DI work in Angular
3. What and Why services?
4. How to create and use a service?
5. Service providers and injectors
6. Injector hierarchy in Angular(root, module and component)

What is Dependency Injection ?

Narasimha

Sr. IT Trainer/Consultant

What is Dependency Injection

- Dependency Injection(DI) is a design pattern and mechanism for creating and delivering some parts of an application to other parts of an application that require them.
- DI is one of the fundamental concepts in Angular.
- Angular supports this design pattern and you can use it in your applications to increase flexibility and modularity.

How does DI work in Angular?

Narasimha

Sr. IT Trainer/Consultant

How does DI work in Angular?

- Two main roles exist in the DI system: dependency consumer and dependency provider.
- Angular facilitates the interaction between dependency consumers and dependency providers using **Injector**.
- When a dependency is requested, the injector checks its registry to see if there is an instance already available there.
- If not, a new instance is created and stored in the registry.

What and Why services?

Narasimha

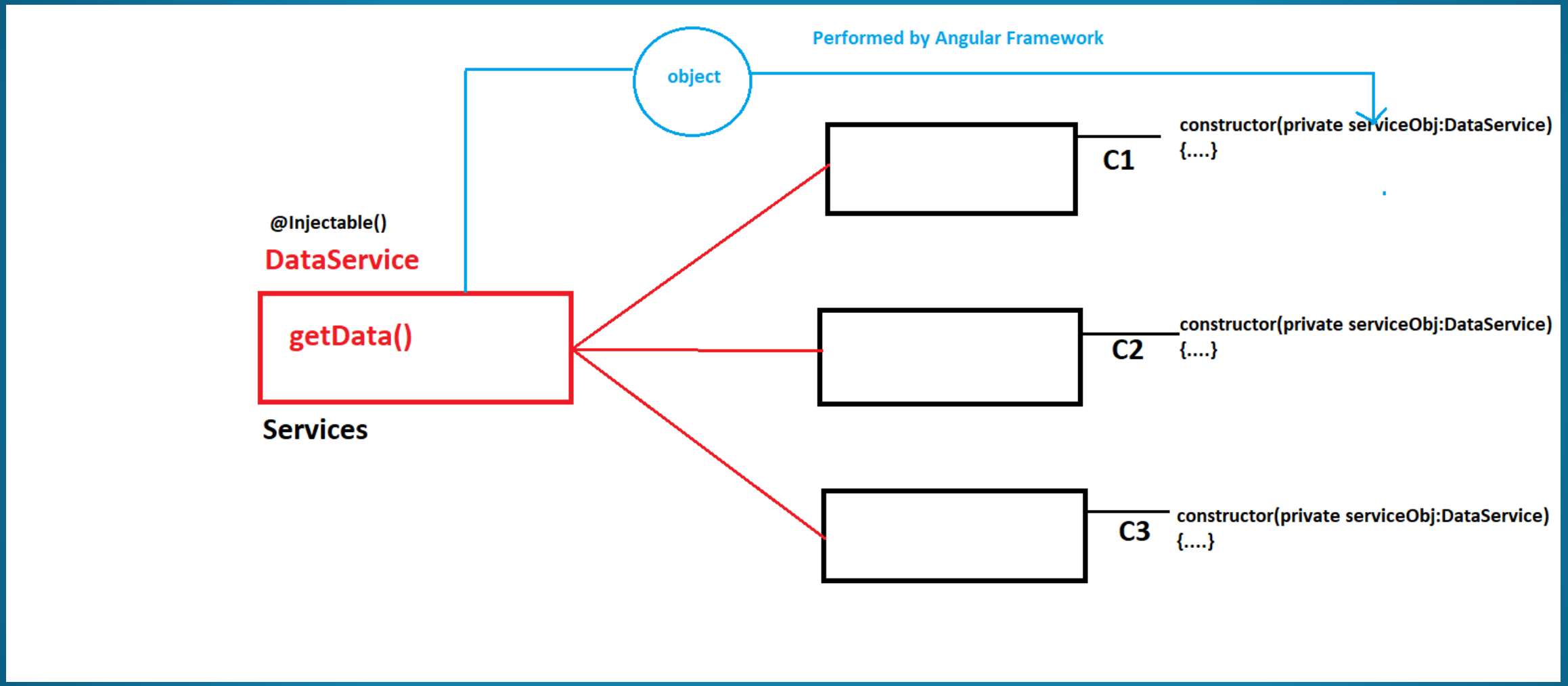
Sr. IT Trainer/Consultant

What and Why services?

- In Angular, dependencies are typically services.
- A service is typically a class with a narrow, well-defined purpose.
- A component is one type of class that can use DI.
- Angular distinguishes components from services to increase modularity and reusability.
- By separating a component's view-related features from other kinds of processing, you can make your component classes lean and efficient.

What and Why services?

- Angular helps you follow these principles by making it easy to factor your application logic into services and make those services available to components through DI.


Performed by Angular Framework
object
constructor(private serviceObj:DataService)
{....}
C1
@Injectable()
DataService
getData()
Services
constructor(private serviceObj:DataService)
{....}
C2
constructor(private serviceObj:DataService)
{....}
C3

How to develop the Services?

Narasimha

Sr. IT Trainer/Consultant

Working with services

Steps

1. Create a service using angular CLI.
2. Add the required logic in service class
3. Provide the services using Providers
4. Inject the service object in component
5. Access the members of services to perform operation.

How to create services?

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

@Injectable({
     providedIn: 'root'
})
export class LogService
{
        log(msg: any) { console.log(msg); }
        error(msg: any) { console.error(msg); }
        warn(msg: any) { console.warn(msg); }
}
```

How to use services?

```
export class AppComponent
{
        constructor(private logger: LogService){ }

        button_click()
        {
                this.logger.log("Hello World");
        }

}
```


Practical
HandsOns

Services Providers and Injectors

Narasimha

Sr. IT Trainer/Consultant

Injector Hierarchy

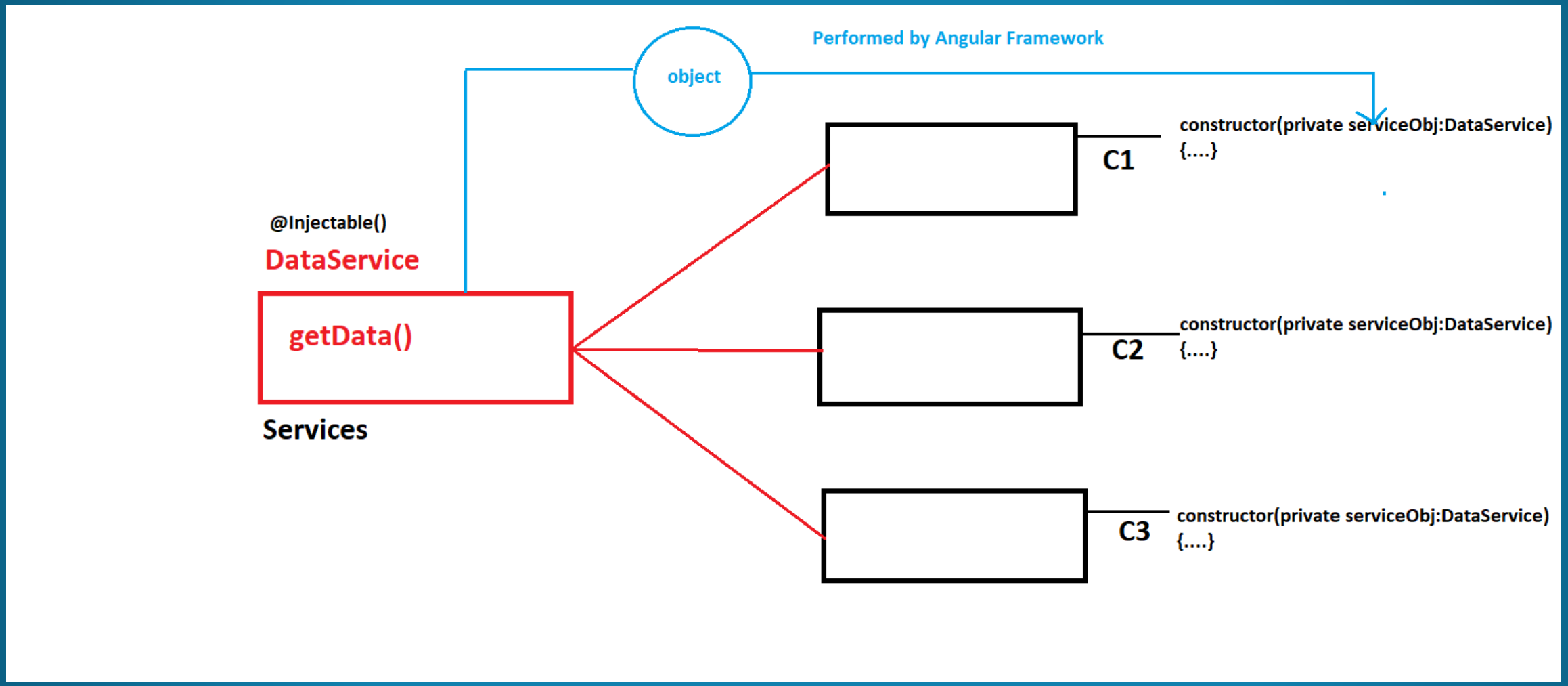
Narasimha

Sr. IT Trainer/Consultant

Injector Hierarchy

There are two injector hierarchies in Angular:

1. ModuleInjector hierarchy—configure a ModuleInjector in this hierarchy using an @NgModule() or @Injectable() providedIn.
2. ElementInjector hierarchy—created implicitly at each DOM element. An ElementInjector is empty by default unless you configure it in the providers property on @Directive() or @Component().


Performed by Angular Framework
object
constructor(private serviceObj:DataService)
{....}
C1
@Injectable()
DataService
getData()
Services
constructor(private serviceObj:DataService)
{....}
C2
constructor(private serviceObj:DataService)
{....}
C3


Practical
HandsOns


Q & A