**Services and DI in Angular:**

**Reference URLs:**

[Introduction to Angular Dependency Injection - TekTutorialsHub](https://www.tektutorialshub.com/angular/angular-dependency-injection/)

[Angular Providers: useClass, useValue, useFactory & useExisting - TekTutorialsHub](https://www.tektutorialshub.com/angular/angular-providers/)

[Create Services In Angular Application (c-sharpcorner.com)](https://www.c-sharpcorner.com/article/create-services-in-angular-application/)

**For Logging Service:** [Angular: Adding Logging in Angular Applications (codemag.com)](https://www.codemag.com/article/1711021/Logging-in-Angular-Applications)

[Angular Providers Example (concretepage.com)](https://www.concretepage.com/angular-2/angular-2-4-providers-example)

[ProvidedIn root, any & platform in Angular - TekTutorialsHub](https://www.tektutorialshub.com/angular/providedin-root-any-platform-in-angular/)

Introduction to Angular Dependency Injection

The Angular dependency injection is now the core part of the [Angular](https://www.tektutorialshub.com/angular-tutorial/). It allows us to inject dependencies into the [Component](https://www.tektutorialshub.com/angular/angular-component/), [Directives](https://www.tektutorialshub.com/angular/angular-directives/), [Pipes](https://www.tektutorialshub.com/angular/angular-pipes/), or [Services](https://www.tektutorialshub.com/angular/angular-services/).

**Dependency Injection (DI) is a technique in which a class receives its dependencies from external sources rather than creating them itself**.

## Benefits of Dependency Injection

### loosely coupled

Component is loosely coupled to the Service. It does not know how to create the Service.  Actually, it does not know anything about the Service. It just works with the Service passed onto it. You can pass any Service. The Component does not care.

### Easier to Test

Component is now easier to Test. Our Component is not dependent on a particular implementation of Service anymore. It will work with any implementation of Service that is passed on to it. You can just create a mockService Class and pass it while testing.

### Reusing the Component

Reusing of the component is becomes easier. Our Component will now work with any Service as long as the interface is honored.

Dependency injection pattern makes our Component testable, maintainable, etc.

## Angular Dependency Injection Framework

Angular Dependency Injection framework implements the Dependency Injection in Angular. It creates & maintains the Dependencies and injects them into the Components, Directives, or Services.

There are five main players in the Angular Dependency injection Framework.

### Consumer

The Consumer is the class (Component, Directive, or Service) that needs the Dependency.

### Dependency

The [Service](https://www.tektutorialshub.com/angular/angular-services/) that we want to in our consumer.

### Injection Token (DI Token)

The [Injection Token](https://www.tektutorialshub.com/angular/injection-token-in-angular/) (DI Token) uniquely identifies a Dependency. We use [DI Token](https://www.tektutorialshub.com/angular/injection-token-in-angular/) when we register dependency

What is Provided

How it is Provided

### Provider

The [Providers](https://www.tektutorialshub.com/angular/angular-providers/) Maintain the list of Dependencies along with their [Injection Token](https://www.tektutorialshub.com/angular/injection-token-in-angular/). It uses the Injection Token is to identify the Dependency.

### Injector

[Injector](https://www.tektutorialshub.com/angular/angular-injector-injectable-inject/) holds the [*Providers*](https://www.tektutorialshub.com/angular/angular-providers/) and is responsible for resolving the dependencies and injecting the instance of the Dependency to the Consumer

The [Injector](https://www.tektutorialshub.com/angular/angular-injector-injectable-inject/) uses Injection Token to search for Dependency in the [Providers](https://www.tektutorialshub.com/angular/angular-providers/). It then creates an instance of the dependency and injects it into the consumer

# Angular Providers: useClass, useValue, useFactory & useExisting

Angular Providers allows us to register classes, functions, or values (dependencies) with the [Angular Dependency Injection system](https://www.tektutorialshub.com/angular/angular-dependency-injection/). The Providers are registered using the **token**. The tokens are used to locate the provider. We can create three types of the token. Type Token, string token & Injection Token. The Provider also tells the [Angular Injector](https://www.tektutorialshub.com/angular/angular-injector-injectable-inject/) how to create the instance of dependency. There are four ways by which you can create the dependency: They are Class Provider (useClass), Value Provider (useValue ), Factory Provider ( useFactory ), and Aliased Class Provider ( useExisting).

## What are Angular Providers

The **Angular Provider** is an instruction (or recipe) that describes how an object for a certain token is created. The **Angular Providers** is an array of such instructions (Provider). Each provider is uniquely identified by a **token** (or DI  Token ) in the Providers Array.

We register the services participating in the [dependency injections](https://www.tektutorialshub.com/angular/angular-dependency-injection/) in the Providers metadata. There are two ways by which we can do it.

1. Register directly in the Providers array of the @NgModule or @Component or in @Directive.
2. Or use the providedIn property of the @[Injectable](https://www.tektutorialshub.com/angular/angular-injector-injectable-inject/) decorator.

The Angular creates an [Injector](https://www.tektutorialshub.com/angular/angular-injector-injectable-inject/) for each component/directive it creates. It also creates a root-level injector, which has the app-level scope. It also creates a Module level Injector for [Lazy Loaded Modules](https://www.tektutorialshub.com/angular/angular-lazy-loading/).

The [Angular Components](https://www.tektutorialshub.com/angular/angular-component/) or [Angular Services](https://www.tektutorialshub.com/angular/angular-services/) declare the dependencies they need in their constructor. The [Injector](https://www.tektutorialshub.com/angular/angular-injector-injectable-inject/) reads the dependencies and looks for the provider in the providers array using the Token. It then instantiates the dependency using the instructions provided by the provider. The Injector then injects the instance of the dependency into the Components/Services.

## Configuring the Angular Provider

To Provide an instance of the dependency, we need to register it in the Providers metadata

In our last tutorial on [Angular Dependency injection](https://www.tektutorialshub.com/angular/angular-dependency-injection/), we registered our ProductService using the Providers arrays as shown below in the @NgModule

|  |  |
| --- | --- |
| 1  2  3 | providers: [ProductService] |

The above is an actual shorthand notation for the following syntax

|  |  |
| --- | --- |
| 1  2  3 | providers :[{ provide: ProductService, useClass: ProductService }] |

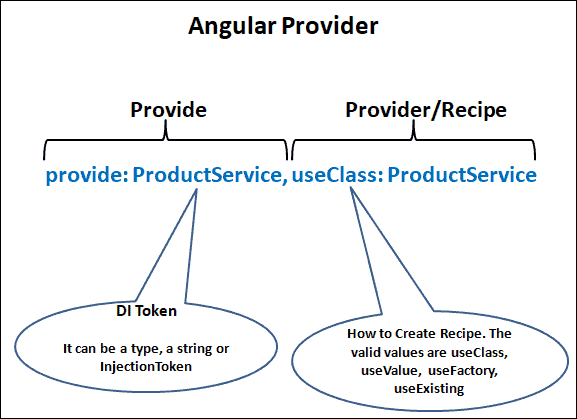
The above syntax has two properties.

### Provide

The first property is Provide holds the [**Token or DI Token**](https://www.tektutorialshub.com/angular/injection-token-in-angular/). The Injector uses the token to locate the provider in the Providers array. The Token can be either a type, a string or an instance of [InjectionToken](https://www.tektutorialshub.com/angular/injection-token-in-angular/).

### Provider

The second property is the Provider definition object. It tells Angular how to create the instance of the dependency. The Angular can create the instance of the dependency in four different ways. It can create a dependency from the existing service class (useClass). It can inject a value, array, or object (useValue). It can use a factory function, which returns the instance of service class or value (useFactory). It can return the instance from an already existing token (useExisting).



## DI Token

The Injector maintains an **internal collection of token-provider** in the Providers array. The token acts as a key to that collection & Injector use that Token (key) to locate the Provider.

The [DI Token](https://www.tektutorialshub.com/angular/injection-token-in-angular/) can be either type, a string or an instance of InjectionToken.

### Type Token

Here the type being injected is used as the token.

For Example, we would like to inject the instance of the ProductService, we will use the ProducService as the token as shown below

|  |  |
| --- | --- |
| 1  2  3 | providers :[{ provide: ProductService, useClass: ProductService }] |

The ProductService is then injected to the component by using the following code.

|  |  |
| --- | --- |
| 1  2  3  4  5 | class ProductComponent {    constructor(private productService : ProductService ) {}  } |

You can keep the same token (ProductService) and change the class to another implementation of the Product service. For Example in the following code, we change it to BetterProductService.

|  |  |
| --- | --- |
| 1  2  3  4 | providers: [      { provide: ProductService, useClass: BetterProductService }, |

Angular does not complain if we use the token again. In the following example token ProductService used twice. In such a situation last to register wins (BetterProductService).

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | providers: [      { provide: ProductService, useClass: ProductService },      { provide: ProductService, useClass: BetterProductService }    ] |

Please refer to the following URL for more information:

[Angular Providers: useClass, useValue, useFactory & useExisting - TekTutorialsHub](https://www.tektutorialshub.com/angular/angular-providers/)

## **What is Service and why do we use it?**

* A service in Angular is a class which contains some functionality that can be reused across the application. A service is a singleton object. Angular services are a mechanism of abstracting shared code and functionality throughout the application.
* Angular Services come as objects which are wired together using dependency injection.
* Angular provides a few inbuilt services. We can also create custom services.

## **Why Services?**

* Services can be used to share the code across components of an application.
* Services can be used to make HTTP requests.

## **Creating a Service**

Create a service class using the following command.

1. ng generate service Article

The above command will create a service class (article.service.ts) as shown below.

1. **import** { Injectable } from '@angular/core';
3. @Injectable({
4. providedIn: 'root'
5. })
6. **export** **class** ArticleService {
8. constructor() { }
9. }

@Injectable() decorator makes the class injectable into application components.https://www.c-sharpcorner.com/CuteSoft_Client/CuteEditor/Images/emidea.gif

## **Providing a Service**

Services can be provided in an Angular applications in any of the following ways:

The first way  to register service is to specify providedIn property using @Injectable decorator. This property is added by default when you generate a service using Angular CLI.

1. **import** { Injectable } from '@angular/core';
3. @Injectable({
4. providedIn: 'root'
5. })
6. **export** **class** ArticleService {
8. constructor() { }
9. }

Line 4: providedIn property registers articleService at the root level (app module).

When the ArticleService is provided at the root level, Angular creates a singleton instance of the service class and injects the same instance into any class that uses this service class. In addition, Angular also optimizes the application if registered through providedIn property by removing the service class if none of the components use it.

There is also a way to limit the scope of the service class by registering it in the providers' property inside @Component decorator. Providers in component decorator and module decorator are independent. Providing a service class inside component creates a separate instance for that component and its nested components.

Add the below code in app.components.ts,

1. **import** { Component } from '@angular/core';
2. **import** { ArticleService } from './article.service';
3. @Component({
4. selector: 'app-root',
5. templateUrl: './app.component.html',
6. styleUrls: ['./app.component.css'],
7. providers : [ArticleService]
8. })
9. **export** **class** AppComponent {
10. title = 'FormsProject';
11. }

Services can also be provided across the application by registering it using providers property in @Ngmodule decorator of any module.

1. **import** { BrowserModule } from '@angular/platform-browser';
2. **import** { NgModule } from '@angular/core';
3. **import** { ReactiveFormsModule } from '@angular/forms';
4. **import** { AppRoutingModule } from './app-routing.module';
5. **import** { AppComponent } from './app.component';
6. **import** {Form, FormsModule} from '@angular/forms';
7. **import** { ArticleFormComponent } from './article-form/article-form.component';
8. **import** { RegistrationFormComponent } from './registration-form/registration-form.component';
9. **import** { ArticleService } from './article.service';
10. @NgModule({
11. declarations: [
12. AppComponent,
13. ArticleFormComponent,
14. RegistrationFormComponent
15. ],
16. imports: [
17. BrowserModule,
18. AppRoutingModule,
19. FormsModule,
20. ReactiveFormsModule
21. ],
22. providers: [ArticleService],
23. bootstrap: [AppComponent]
24. })
25. **export** **class** AppModule { }

Line 22: When the service class is added in the providers property of the root module, all the directives and components will have access to the same instance of the service.

## **Injecting a Service**

The only way to inject a service into a component/directive or any other class is through a constructor. Add a constructor in a component class with service class as an argument as shown below,

Here, ArticleService will be injected into the component through constructor injection by the framework. https://www.c-sharpcorner.com/CuteSoft_Client/CuteEditor/Images/emidea.gif

1. **import** { Component } from '@angular/core';
2. **import** { ArticleService } from './article.service';
3. @Component({
4. selector: 'app-root',
5. templateUrl: './app.component.html',
6. styleUrls: ['./app.component.css'],
7. providers : [ArticleService]
8. })
9. **export** **class** AppComponent {
10. title = 'FormsProject';
12. constructor(**private** articleService: ArticleService){ }
13. }

**Injecting Service in a Non-Root Module (in Module other than App.Module):**

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

**import { UserModule } from './user.module';**

**@Injectable({**

**providedIn: UserModule,**

**})**

**export class UserService {**

**}**

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

**import { UserService } from './user.service';**

**@NgModule({**

**providers: [UserService],**

**})**

**export class UserModule {**

**}**

**Problem Statement**

Create an Article Component which fetches article details like id, name and displays them on the page in a list format. Store the article details in an array and fetch the data using a custom service.

## **Demosteps**

Create ArticleComponent by using the following CLI command

1. ng generate component Article

Create a file with name Article.ts under book folder and add the following code.

1. **export** **class** Article {
2. id: number;
3. name: string;
4. }

Create a file with name Article-data.ts under book folder and add the following code.

1. **import** {Article} from './Article';
2. **export** **var** ARTICLES: Article[] = [
3. { "id": 1, "name": "Angular Basic" },
4. { "id": 2, "name": "Template in Angular" },
5. { "id": 3, "name": "Nested component" },
6. { "id": 4, "name": "Reactive component" },
7. { "id": 5, "name": "Change detection technique" }
8. ];

Create a service called ArticleService under book folder using the following CLI command,

1. ng generate service Article

Add the following code in **a**rticle.service.ts

1. **import** { Injectable } from '@angular/core';
2. **import** {ARTICLES} from './Article-data';
3. **import** {Article} from './Article';
5. @Injectable({
6. providedIn: 'root'
7. })
8. **export** **class** ArticleService {
10. getArticles ()
11. {
12. **return** ARTICLES;
13. }
14. }

Add the following code in article.component.ts file

1. **import** { Component, OnInit } from '@angular/core';
2. **import** {ArticleService} from './article.service';
3. **import** { Article } from './Article';
4. @Component({
5. selector: 'app-article',
6. templateUrl: './article.component.html',
7. styleUrls: ['./article.component.css']
8. })
9. **export** **class** ArticleComponent **implements** OnInit {
10. articles : Article[];
12. constructor(**private** articelService : ArticleService) { }
13. getArticles()
14. {
15. **this**.articles=**this**.articelService.getArticles()
16. }
17. ngOnInit() {
18. **this**.getArticles()
19. }
20. }

Write the below-given code in article.component.html

1. <h2>My Articles</h2>
2. <ul **class**="Articles">
3. <li \*ngFor="let article of Articles">
4. <span **class**="badge">{{article.id}}</span> {{article.name}}
5. </li>
6. </ul>

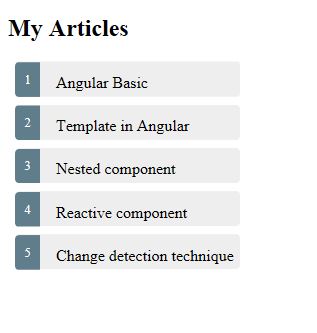
Add the following code in article.component.css which has styles for books.

1. .Articles {
2. margin: 0 0 2em 0;
3. list-style-type: none;
4. padding: 0;
5. width: 15em;
6. }
7. .Articles li {
8. cursor: pointer;
9. position: relative;
10. left: 0;
11. background-color: #EEE;
12. margin: .5em;
13. padding: .3em 0;
14. height: 1.6em;
15. border-radius: 4px;
16. }
17. .Articles li:hover {
18. color: #607D8B;
19. background-color: #DDD;
20. left: .1em;
21. }
22. .Articles .badge {
23. display: inline-block;
24. font-size: small;
25. color: white;
26. padding: 0.8em 0.7em 0 0.7em;
27. background-color: #607D8B;
28. line-height: 1em;
29. position: relative;
30. left: -1px;
31. top: -4px;
32. height: 1.8em;
33. margin-right: .8em;
34. border-radius: 4px 0 0 4px;
35. }

Add the following code in app.component.html.

1. <app-article></app-article>

Save the files and check the output in the browser,



Please refer the following URL for For useClass, useExisting, useValue and useFactory:

[Angular Providers Example (concretepage.com)](https://www.concretepage.com/angular-2/angular-2-4-providers-example)