Dependency Injection in Angular

**Dependency Injection:**

In software engineering, dependency injection is a technique whereby one object supplies the dependencies of another object. In other words, we can say that, it is a coding pattern in which classes receives their dependencies from external sources rather than creating them itself.

Dependency Injection is the heart of Angular Applications. The Dependency Injection in Angular is a combination of two terms i.e. Dependency and Injection.

**Dependency**: Dependency is an object or service that is going to be used by another object.

**Injections**: It is a process of passing the dependency object to the dependent object. It creates a new instance of the class along with its require dependencies.

**Understanding Dependency Injection in Angular with an Example:**

Let us understand dependency injection in Angular with an example. First create a service with the name **StudentService**and then copy and paste the following code in it.

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

@Injectable**()**

**export** **class** StudentService **{**

getTitle**()**

**{**

**return** "Dependency Injection in Angular";

**}**

getStudents**()**: **any[]** **{**

**return** **[**

**{**

ID: 'std101', FirstName: 'Preety', LastName: 'Tiwary',

Branch: 'CSE', DOB: '29/02/1988', Gender: 'Female'

**}**,

**{**

ID: 'std102', FirstName: 'Anurag', LastName: 'Mohanty',

Branch: 'ETC', DOB: '23/05/1989', Gender: 'Male'

**}**,

**{**

ID: 'std103', FirstName: 'Priyanka', LastName: 'Dewangan',

Branch: 'CSE', DOB: '24/07/1992', Gender: 'Female'

**}**,

**]**;

**}**

**}**

**Modify app.component.tsfile:**

Open **app.component.ts**file and then copy and paste the following code in it.

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

**import** **{**StudentService**}** from './student.service';

@Component**({**

selector: 'app-root',

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

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

providers:**[**StudentService**]**

**})**

**export** **class** AppComponent **{**

students: **any[]**;

pageTitle: **string**;

**private** \_studentService: StudentService;

**constructor(**studentService: StudentService**)** **{**

this.\_studentService = studentService;

**}**

ngOnInit**()** **{**

this.students = this.\_studentService.getStudents**()**;

this.pageTitle = this.\_studentService.getTitle**()**;

**}**

**}**

**Modify app.component.html file:**

Open **app.component.html**file and then copy and paste the following code in it.

**<h2>**{{pageTitle}}**</h2>**

**<table>**

**<thead>**

**<tr>**

**<th>**ID**</th>**

**<th>**First Name**</th>**

**<th>**Last Name**</th>**

**<th>**Branch**</th>**

**<th>**DOB**</th>**

**<th>**Gender**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr** \*ngFor='let student of students'**>**

**<td>**{{student.ID}}**</td>**

**<td>**{{student.FirstName}}**</td>**

**<td>**{{student.LastName}}**</td>**

**<td>**{{student.Branch}}**</td>**

**<td>**{{student.DOB}}**</td>**

**<td>**{{student.Gender}}**</td>**

**</tr>**

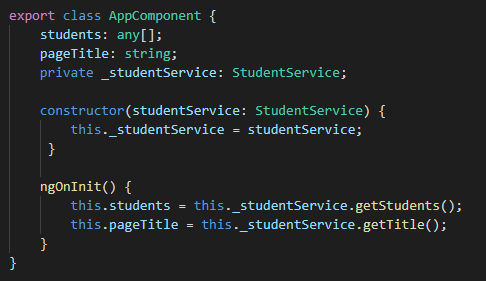
**</tbody>**

**</table>**

Now run the application and you should see the output as expected.

**Understanding the code:**

Now let us proceed and try to understand dependency injection with this example. Please have a look at the following piece of code of AppComponent class.



Here, in the AppComponent class, we need an instance of StudentService to call the the getStudents() and getTitle() method of Student Service to get the list of students and the title of the page which is required by App component.

But if you look at the above AppComponent code, we are not creating an instance of StudentService. Here, we declared a private field \_studentService of type StudentService. The constructor also has a parameter studentService of type StudentService. Then the constructor initializing the private class field \_studentService with it’s parameter studentService .

Then we are using this private field \_studentService to call the Student Service methods getStudents() and getTitle().

Now, the question that should comes to your mind is, how are we getting an instance of the StudentService class.

**How are we getting an instance of the StudentService class?**

From the code of AppComponent class we can see that the constructor is provided with an instance of StudentService class, and then the constructor is assigning that instance to the private field \_studentService.

**Who is creating and providing the instance to the constructor?**

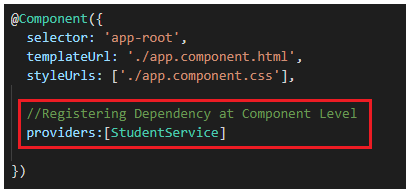
The answer is Angular Injector. When an instance of AppComponent class is created, the angular injector creates an instance of the StudentService class and provides it to the AppComponent constructor. The constructor then assigns that instance to the private field \_studentService. We then use this private field \_studentService to call the StudentService methods getStudents() and getTitle().

**How does the angular injector knows about StudentService?**

For the Angular injector to be able to create and provide an instance of StudentService , first we need to register the StudentService with the Angular Injector. We register a service with the angular injector by using the providers property of **@Component** decorator or **@NgModule** decorator. We already know we decorate an angular component with @Component decorator and an angular module with @NgModule decorator.

**At Component Level:**

If you are registering a service using the providers property of the @Component decorator then you are registering the service with an angular injector at the component level. The service is then available to that component and all of it’s children. The syntax to register dependency at component level is given below. Here, you need to register the service within the Providers array of Component decorator.



**At Module Level:**

On the other hand if you register the service using the providers property of the @NgModule decorator then you are registering the service with an angular injector at the module level which is the root injector. The service registered with the root injector is then available to all the component across the entire application. The syntax to register at app level is given below. As you can see, you need to declare the service within the Providers array of your root module i.e. AppModule.



So, in our example, AppComponent has a dependency on StudentService. The AppComponent receives the dependency instance (i.e StudentService instance) from the the external source (i.e the angular injector) rather than creating the instance itself.