breaking thought barriers

Angular Training

Session -22

Angular Service

Angular Services

- Service is a class that has the purpose of Providing a Service to a Component, Directive, or to another Service.
- A service can contain a value or function or combination of both.
- The Service may be fetching data from the back end, running a business logic, etc
- The Services in angular are injected into the application using the dependency injection mechanism.

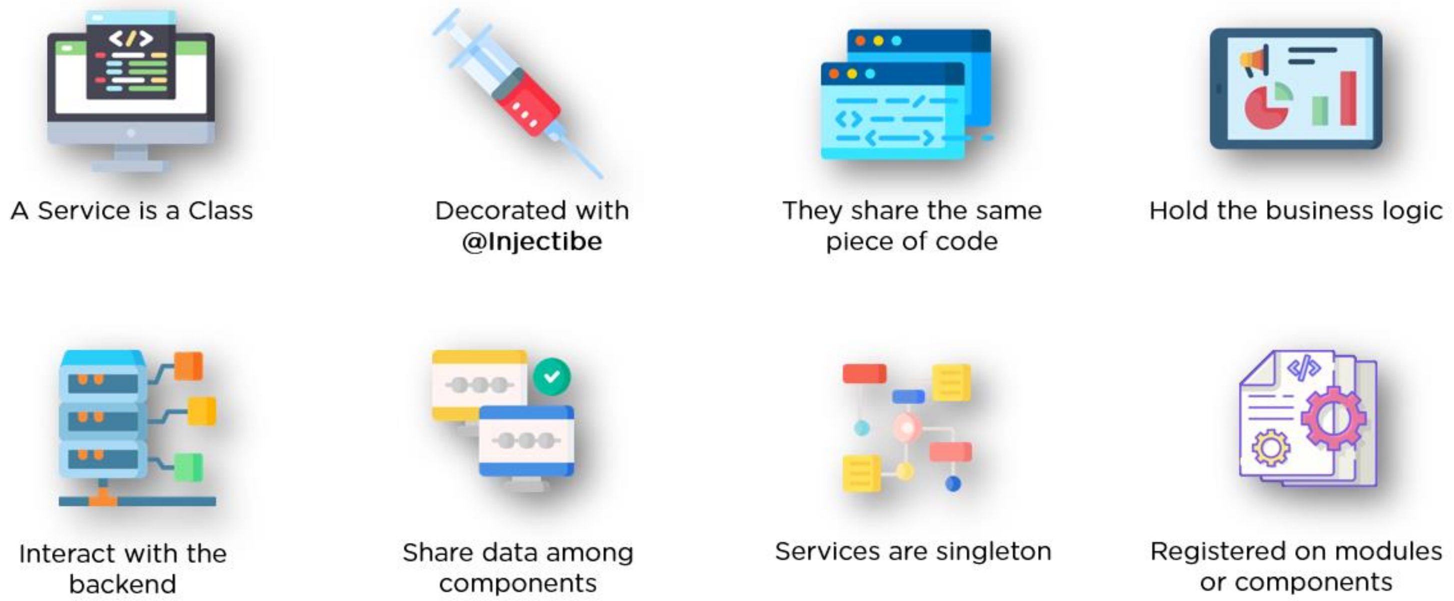
Why do we need a service in Angular?

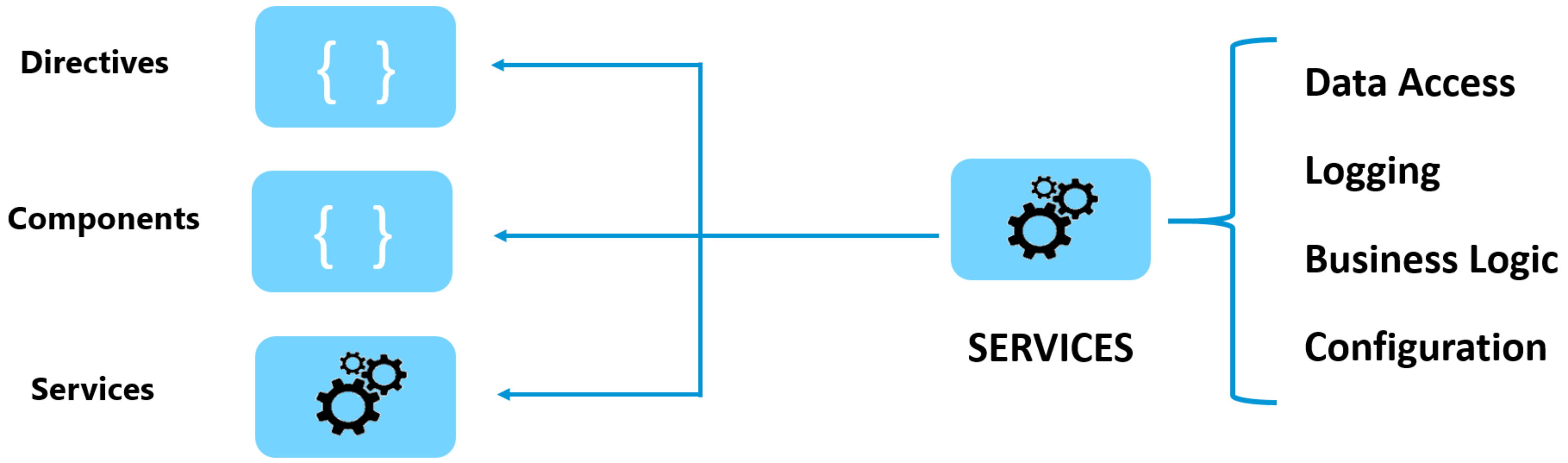
- Whenever you need to reuse the same data and logic across multiple components of your application, then you need to go for angular service.
- The logic or data is implemented in a services and the service can be used across multiple components of your angular application. So, services is a mechanism used to share the functionality between the components.

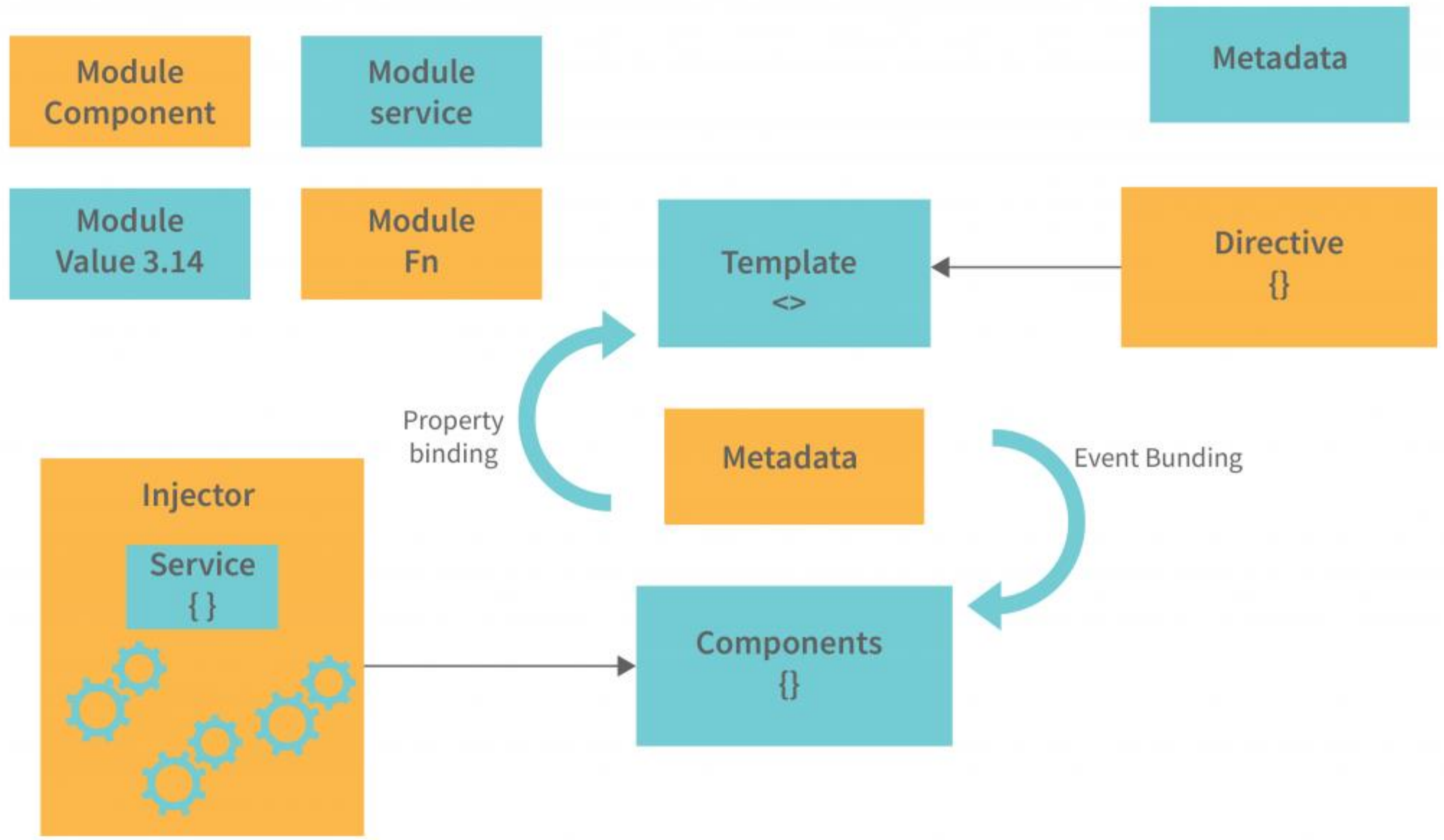
Advantageous of Angular Service ?

1. Services are easier to test.
2. They are easier to Debug.
3. We can reuse the service at many places.

Without Angular Services, we would have to repeat the same logic or code in multiple components wherever we want this code or logic.


A Service is a Class
Decorated with @Injectibe
They share the same piece of code
Hold the business logic
Interact with the backend
Share data among components
Services are singleton
Registered on modules or components


Directives
Components
Services
SERVICES
Data Access
Logging
Business Logic
Configuration


Module Component
Module service
Metadata
Module Value 3.14
Module Fn
Template <>
Directive {}
Property binding
Metadata
Event Bunding
Injector
Service { }
Components {}

How to create a Service in Angular

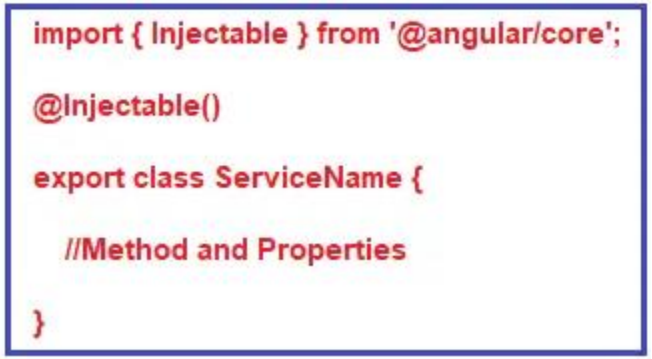
Step1**. Creating Angular Service**

```
ng generate service student
```

The angular service is composed of three things.

- We need to create an export class
- We need to decorate that class with @Injectable decorator
- We need to import the Injectable decorator from the angular core library.

The syntax to create angular service is



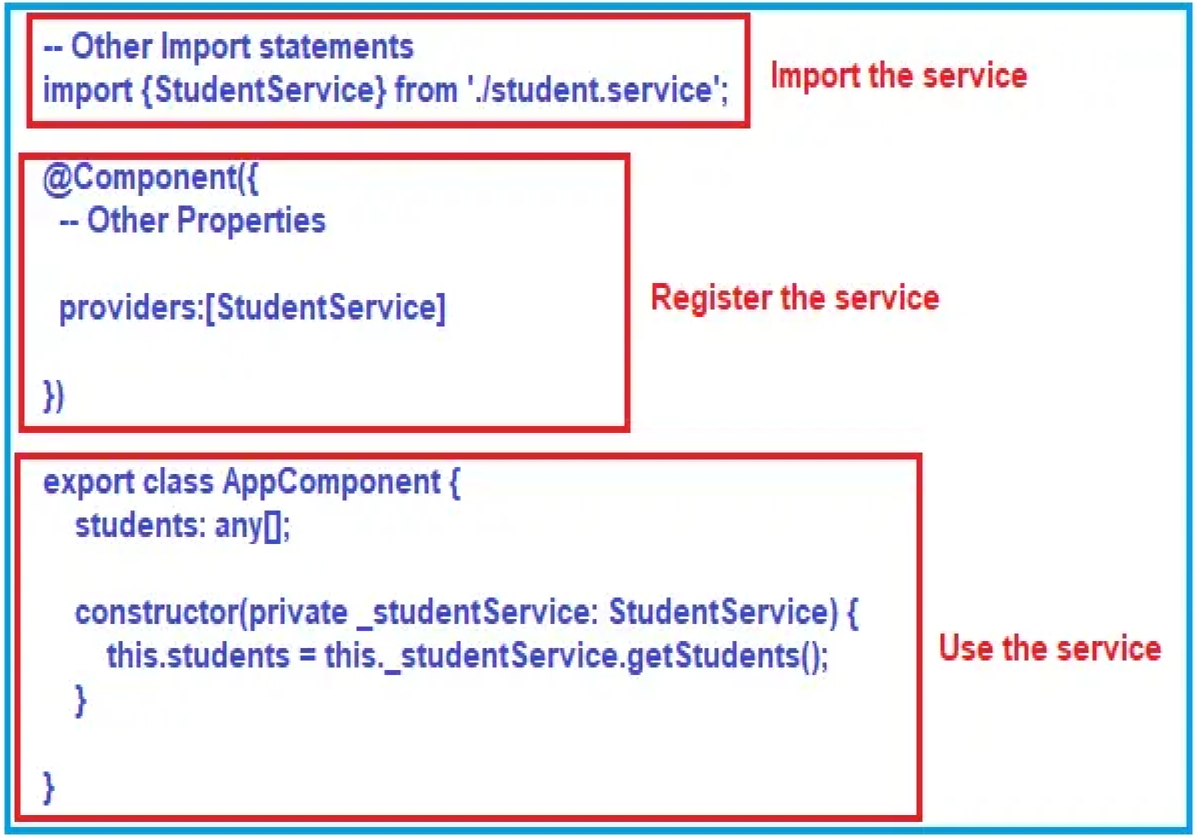
```
import { Injectable } from '@angular/core';

@Injectable()

export class ServiceName {

   //Method and Properties

}
```

```
import { Injectable } from '@angular/core';
@Injectable()
export class StudentService {
  getStudents(): any[] {
    return [
     {
      ID: 'std101', FirstName: 'Preety', LastName: 'Tiwary',
      Branch: 'CSE', DOB: '29/02/1988', Gender: 'Female'
    },
     {
      ID: 'std103', FirstName: 'Priyanka', LastName: 'Dewangan',
      Branch: 'CSE', DOB: '24/07/1992', Gender: 'Female'
    },
    {
      ID: 'std104', FirstName: 'Hina', LastName: 'Sharma',
      Branch: 'ETC', DOB: '19/08/1990', Gender: 'Female'
    },
    {
      ID: 'std105', FirstName: 'Sambit', LastName: 'Satapathy',
      Branch: 'CSE', DOB: '12/94/1991', Gender: 'Male'
    }
    ];
  }
}
```

Note: The @Injectable() decorator in angular is used to inject other dependencies into the service. At the moment our service does not have any other dependencies, so, yo can remove the @Injectable() decorator and the service should works. However, the Angular Team recommends to always use @Injectable()

Step2: Using the Service in Angular

It is a three step process.
a. Import the service
b. register the service
c. use the service.



```
-- Other Import statements
import {StudentService} from './student.service';
```

Import the service

```
@Component({
  -- Other Properties

  providers:[StudentService]

})
```

Register the service

```
export class AppComponent {
    students: any[];

    constructor(private _studentService: StudentService) {
        this.students = this._studentService.getStudents();
    }

}
```

Use the service

Using ngOnInit() life cycle hook to call the getStudents() service Method

```
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[];

    constructor(private _studentService: StudentService) { }

    ngOnInit() {
        this.students = this._studentService.getStudents();
    }
}
```

Difference between constructor and ngOnInit

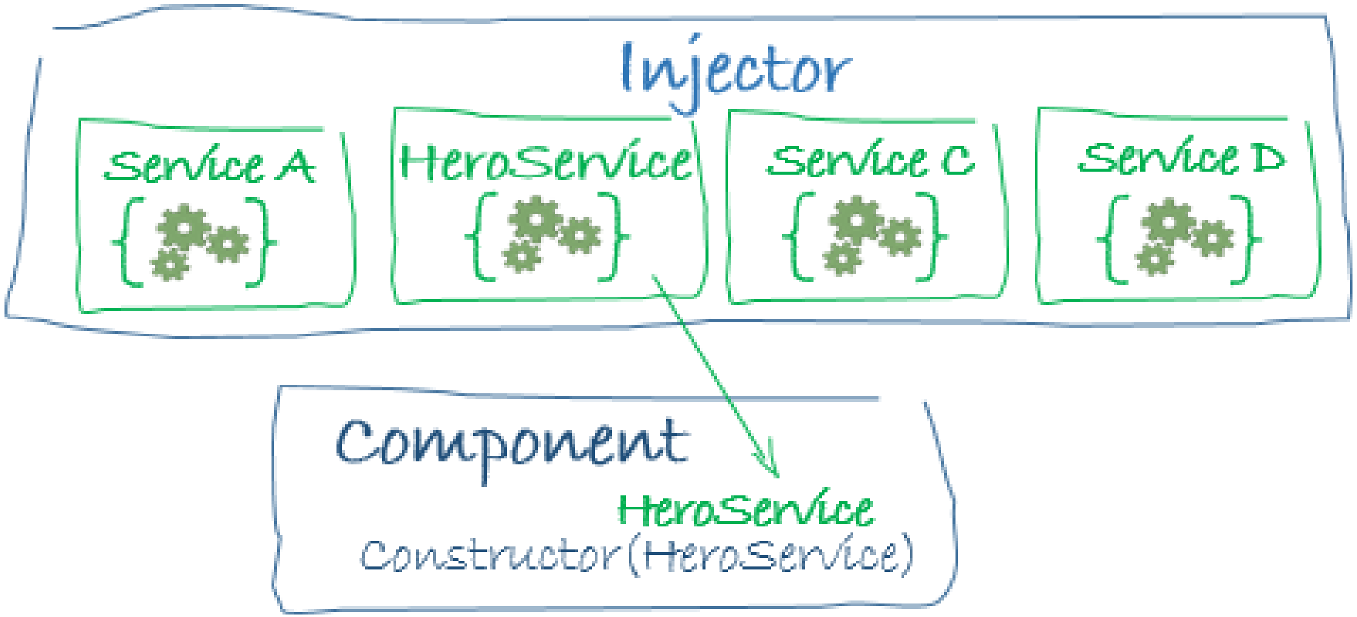
- Whenever you create an instance of a class, the class constructor is automatically called. Like other programming languages, the class constructor in angular is also used to initialize the members of the class and it’s sub classes.

- The ngOnInit is a life cycle hook method provided by Angular which is called after the constructor and is generally used to perform tasks related to Angular bindings.

- For example, ngOnInit is the right place to call a service method to fetch data from a remote server.

- We can also do the same using a class constructor, but the general rule of thumb is, tasks that are time consuming should use ngOnInit instead of the constructor. As fetching data from a remote server is time consuming, the better place for calling the service method is ngOnInit.

Dependency Injection



Dependency Injection in Angular

- 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.

What Is Dependency Injection

- Dependency injection is what makes a class independent of its dependencies.
- Dependency injection enables the creation of dependent objects outside of a class while providing those very objects to a class in numerous ways.

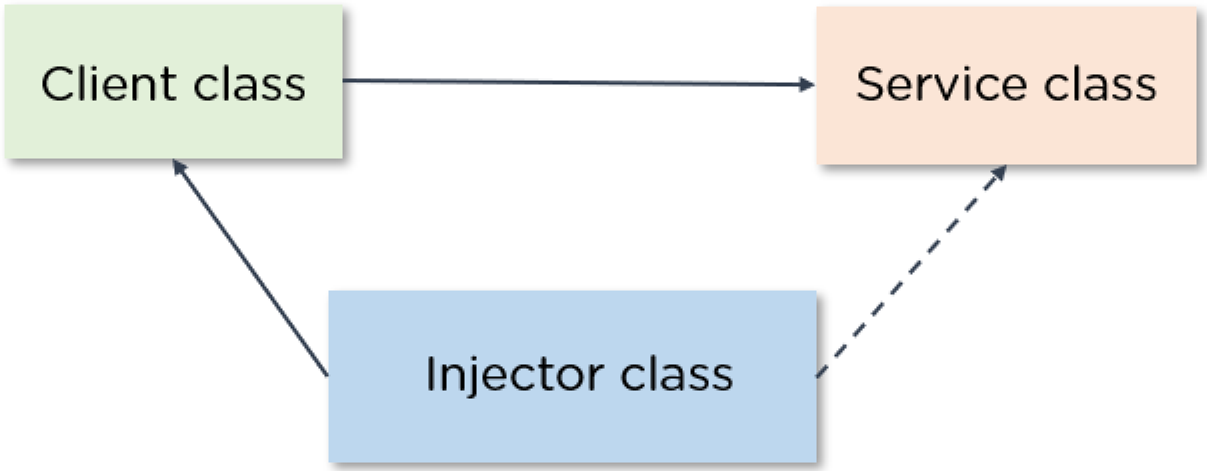
“Delegate creation of instance task to a global authority and just ask for the instance when you want to use it”.

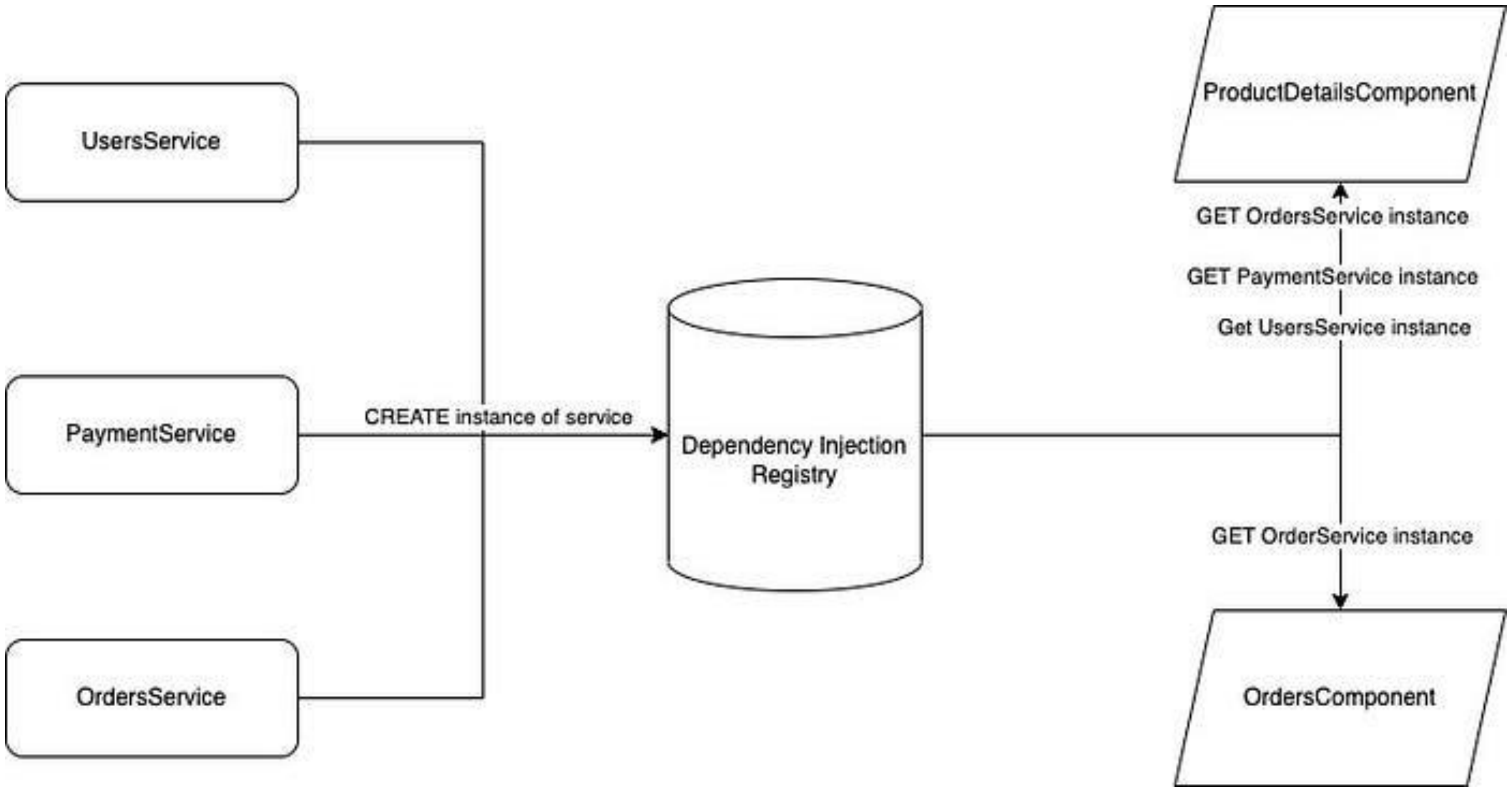
There are three types of classes, they are:

1. Client Class - This is the dependent class, which depends on the service class.

2. Service Class - Class that provides the service to the client class.

3. Injector Class - Injects the service class object into the client class.

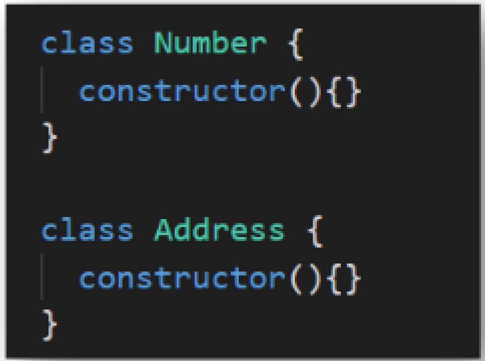

Client class
Service class
Injector class


UsersService
PaymentService
OrdersService
CREATE instance of service
Dependency Injection Registry
ProductDetailsComponent
GET OrdersService instance
GET PaymentService instance
Get UsersService instance
GET OrderService instance
OrdersComponent

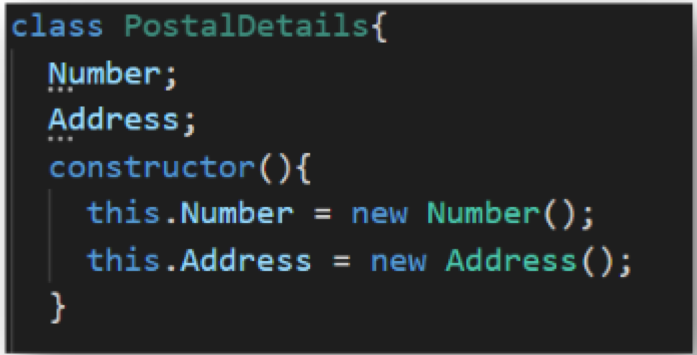
Advantages of Dependency Injection

- Dependency Injection helps in Unit testing.
- Boilerplate code is reduced, as initializing of dependencies is done by the injector component.
- Extending the application becomes more manageable.
- It helps to enable loose coupling, which is essential in application programming.

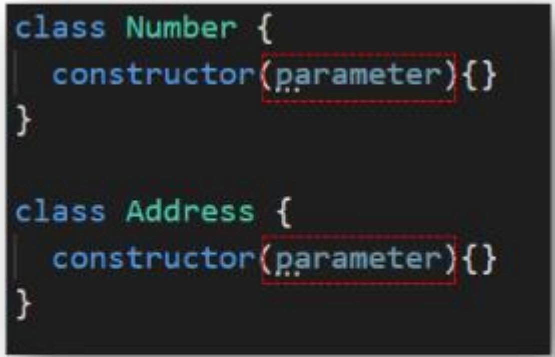
The Drawbacks of not using Dependency Injection



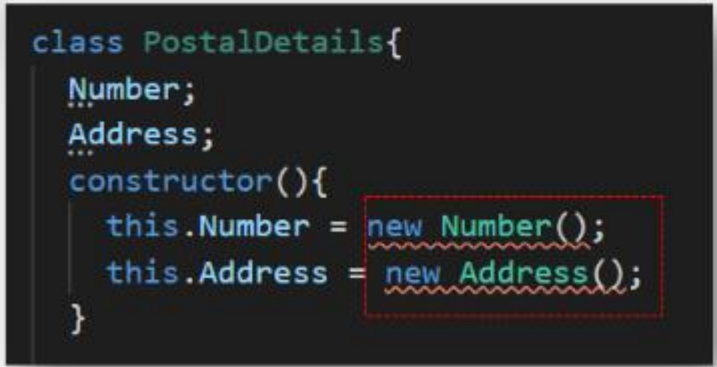
```
class Number {
  constructor(){}
}

class Address {
  constructor(){}
}
```



```
class PostalDetails{
  Number;
  Address;
  constructor(){
    this.Number = new Number();
    this.Address = new Address();
  }
```



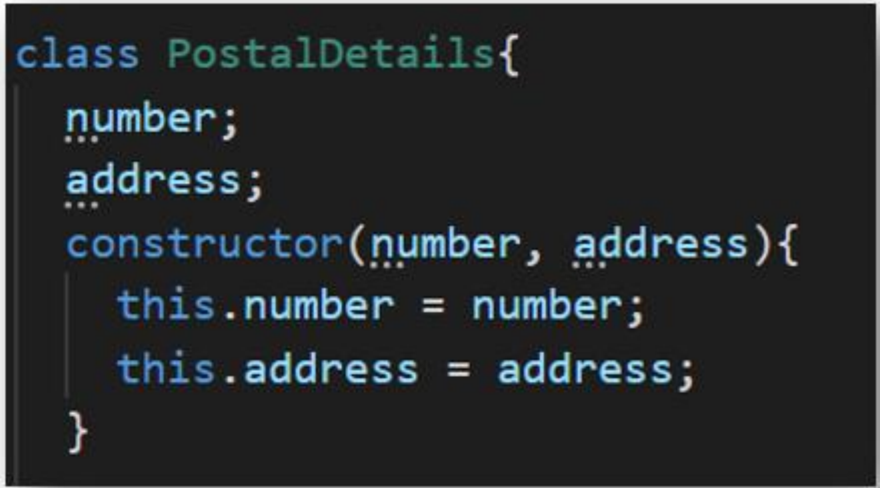
```
class Number {
  constructor(parameter){}
}

class Address {
  constructor(parameter){}
}
```



```
class PostalDetails{
  Number;
  Address;
  constructor(){
    this.Number = new Number();
    this.Address = new Address();
  }
```

Dependency Injection as a Design Pattern

- DI is a coding pattern where a class receives its dependencies from an external source rather than creating them itself.



```
class PostalDetails{
  number;
  address;
  constructor(number, address){
    this.number = number;
    this.address = address;
  }
```

Injecting Services into Components to Display a List.

```
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: 'std103', FirstName: 'Priyanka', LastName: 'Dewangan',
    Branch: 'CSE', DOB: '24/07/1992', Gender: 'Female'
  },
  ];
 }
}
```

```
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();
    }
}
```

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.

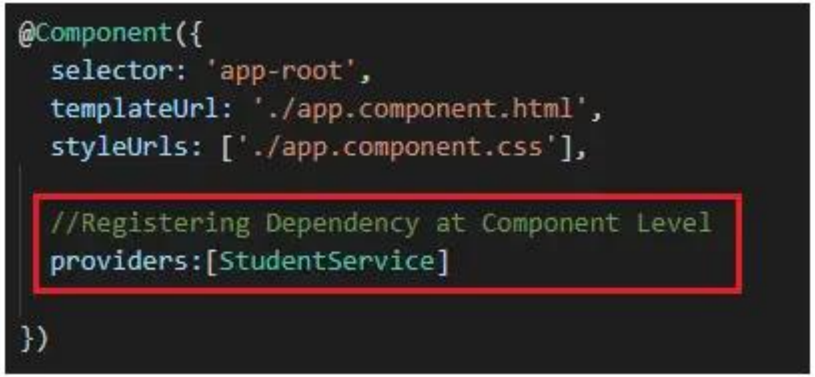
- The constructor then assigns that instance to the private field _studentService. We then use this private field _studentService to call the StudentService methods getStudents() and getTitle()

How does the angular injector knows about Service?

- For the Angular injector to be able to create and provide an instance of Service class , first we need to register the Service class with the Angular Injector.

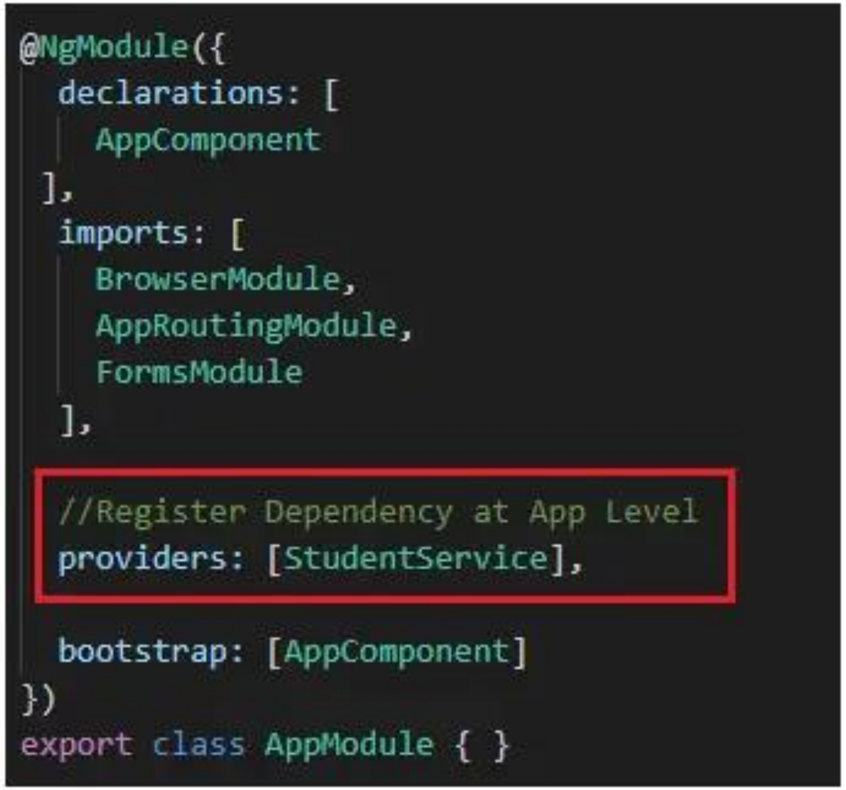
- We register a service with the angular injector by using the providers property of @Component decorator or @NgModule decorator.

At Component Level:



```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],

  //Registering Dependency at Component Level
  providers:[StudentService]

})
```

At Module Level:



```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],

  //Register Dependency at App Level
  providers: [StudentService],

  bootstrap: [AppComponent]
})
export class AppModule { }
```

Benefits of Dependency Injection

loosely coupled

Easier to Test

Reusing the Component

Angular Dependency Injection Framework

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

1. Consumer

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

2. Dependency

The Service that we want to in our consumer.

3. Injection Token (DI Token)

The Injection Token (DI Token) uniquely identifies a Dependency. We use DI Token when we register dependency

4. Provider

The Providers Maintain the list of Dependencies along with their Injection Token. It uses the Injection Token is to identify the Dependency.

5. Injector

Injector holds the Providers and is responsible for resolving the dependencies and injecting the instance of the Dependency to the Consumer

The Injector uses Injection Token to search for Dependency in the Providers. It then creates an instance of the dependency and injects it into the consumer

Add Bootstrap Library
In
Angular

Bootstrap is a popular front-end framework for building responsive web applications, and we can integrate it into an Angular application to quickly create stylish and responsive user interfaces.

Steps to install and use Bootstrap in an Angular project:

1. Install Bootstrap:

```
npm install bootstrap
```

2. Import Bootstrap CSS:

```
/* Add this line at the top of styles.css */
@import "~bootstrap/dist/css/bootstrap.css";
```

3. Add Bootstrap JavaScript (Optional):

If you plan to use Bootstrap's JavaScript components, such as modals or carousels, you can import Bootstrap's JavaScript files into your project.

Angular.json

```
"scripts": [
  "node_modules/bootstrap/dist/js/bootstrap.js"
]
```

4. Use Bootstrap Components:

```
<!-- Example of using Bootstrap components in an Angular template -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">My Angular Bootstrap App</a>
  <!-- Add other Bootstrap components here -->
</nav>
```

Thank You