Introduction to Dependency Injection in Angular

General concept of DI

- Dependency Injection is a system to make parts (class and object) of our program accessible to other parts of the program - and we can canfigure how that happens
- The major benefit of using Dependency Injection is that the client commponent needn't be aware of how to create the dependencies. All the client component need to know is hoe to interact with those dependencies

How DI works in Angular

- Register the "dependency" with Angular
- Describe how the dependency will be injected
- Inject the dependency
- Usually, we may want to have only one instance of the class to be injected, that is, a Singleton.

Dependency Injection Parts

- Dependency Injection in Angular has three pieces:
- the Provider (also often referred to as a binding) maps a token (that can be a string or a class) to a list of dependencies. It tells Angular how to create an object, given a token
- the Injector that holds a set of bindings and is responsible for resolving dependencies and injecting them when creating objects
- the Dependency that is what's being injected

Providing dependencies with NgModule

- The typical way of using DI in Angular is by following these two steps:
  - use NgModule to register what we'll inject - these are called providers and
  - use decorators (generally on a constructor) to specify what we're injecting

An example of DI: user-demo.module.ts

```
import {NgModule} from '@angular/core';
import {CommonModule} from '@angular/common';

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

@NgModule({
    imports: [
        CommonModule
    ],
    providers: [
        UserService
    ],
    declarations: []
})
export class UserDemoModule {}
```

An example of DI: user-demo.component.ts

```
import {Component, OnInit} from '@angular/core';

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

@Component({
    selector: 'app-user-demo',
    templateUrl: './user-demo.component.html',
    styleUrls: ['./user-demo.component.css']
})
export class UserDemoComponent {
    userName: string;

    constructor(private userService: UserService) {}

    signIn(): void {...}
}
```

Providers

There are several ways we can configure resolving injected dependencies in Angular. For instance we can:

- Inject a (singleton) instance of a class
- Inject a value
- Call any function and inject the return value of that function

DI using a Class

- Singleton instance of a class is probably the most common type of injection
- This tells Angular that we want to provide a singleton instance of the class whenever the class is injected
- In this case, we need a token to identify the injection and the class to inject
- In this case the injector will create a singleton behifn the scenes and return the same instance every time we inject it.

DI using a Value

- Another way we can use DI is to provide a value, much like we might use a global constant
- In this case, in the NgModule will be used the:

```
providers: [{provide: "API_URL", useValue: "http://my.api.com/v1"}]
```

- and then use the @Inject() decorator like this:

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

export class AnalyticsDemoComponent {
    constructor (@Inject("API_URL") apiUrl: string) {}
}
```

DI using a Factory

- In some case, the dependency doesn't have any argument required for the constructor. But what happens if a service's constructor requires arguments? We can implement this by using a factory which is a function that can return any object when injected

```
providers: [
    {provide: AnalyticsService, useFactory: () => ...}
]
```

- useFactory takes a function and whatever this function returns will be injected
- Factory dependencies: sometimes also the factory function will have some dependencies of its own.

Summary

To review, when writing our apps there are thhree steps we need to take in order to perform an injection:

- Create the dependency (e.g. the service class)
- Configure the injection (i.e. register the injection with Angular in our NgModule)
- Declare the dependencies on the receiving component

The first thing you shold do is create the service class, that is what is called the injectable, because it is the thing that our components will receive via the injection.

Then if you need an injectable inside a class, you can inject the dependndcy into a function (often a constructor) and Angular's dependency injection framework will locate it and provide it to you.