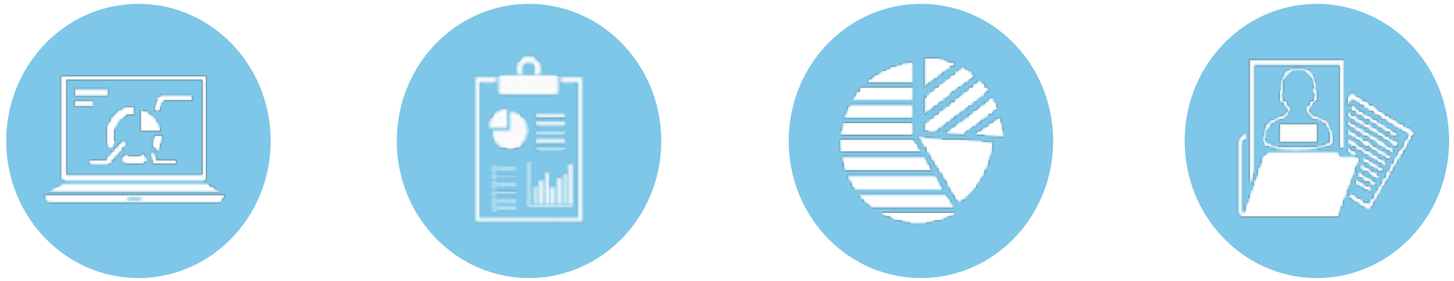
Angular 2

Lesson 6—Dependency Injection and Service



Learning Objectives

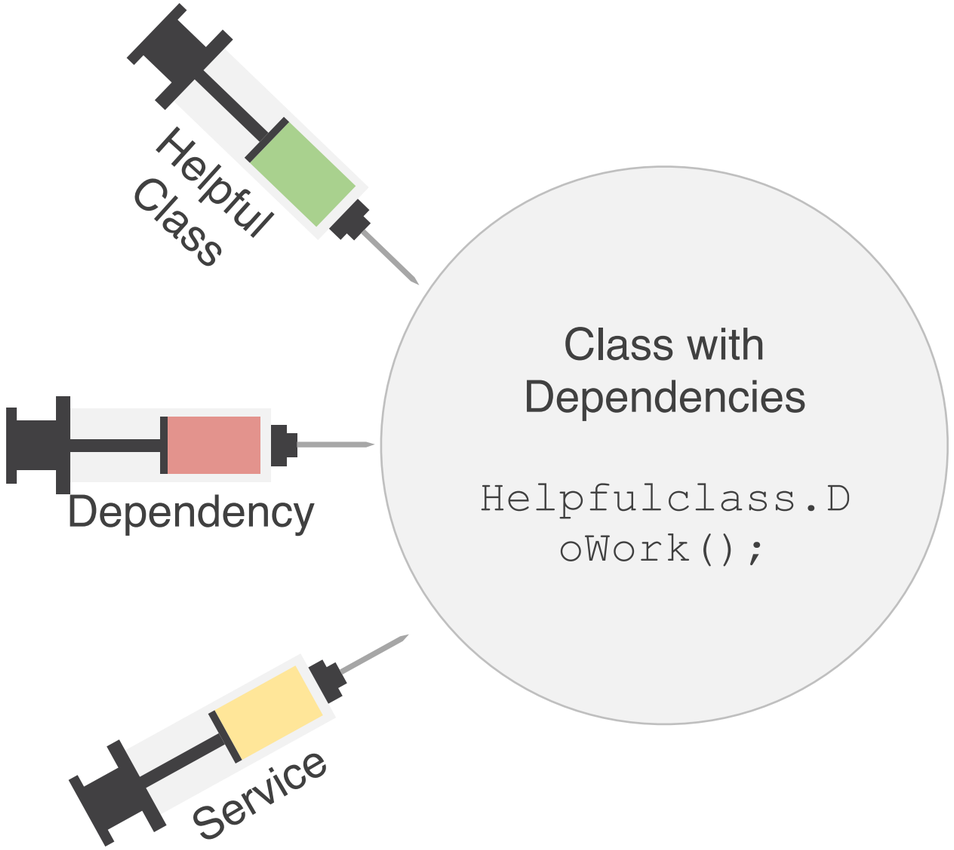
- Understand Dependency Injection (DI)
- Understand DI Application Programming Interface
- Explain a Service
- Describe how to create a Service

Dependency Injection and Service

Topic 1—Dependency Injection

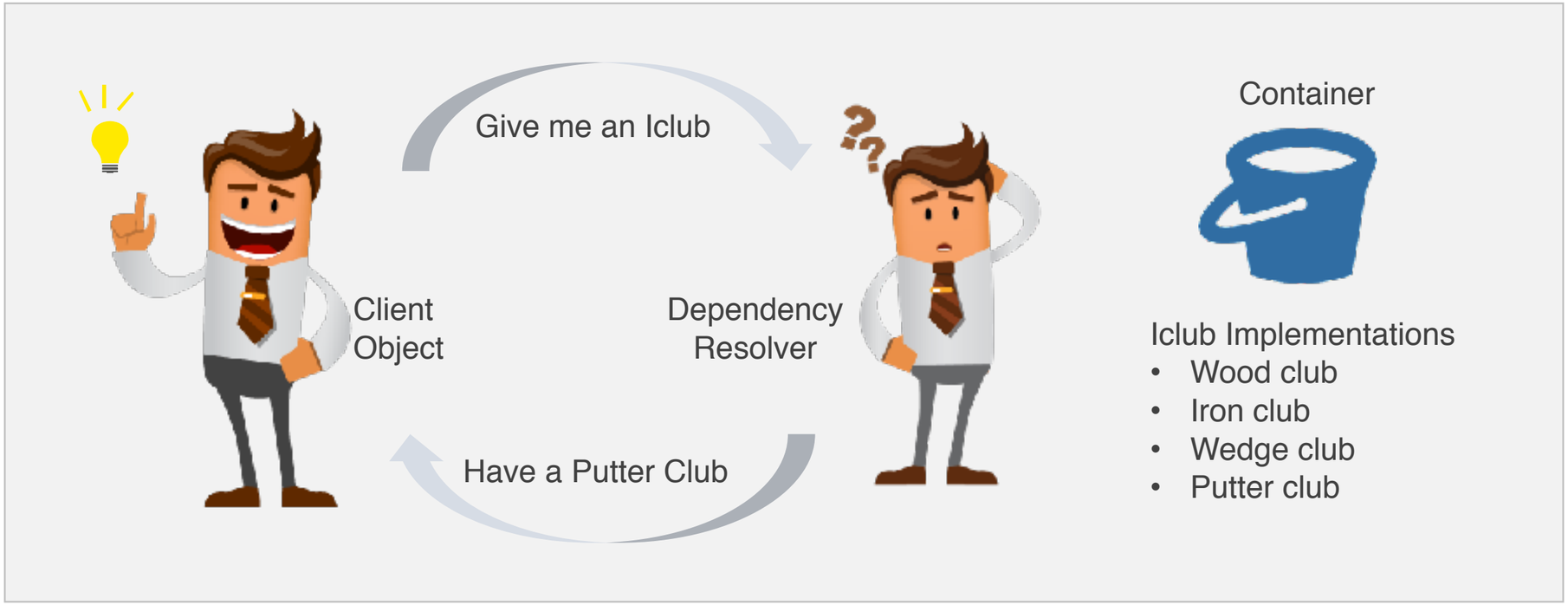
Why Dependency Injection (DI)?

DI is a design pattern that passes an object as a dependency to different components across the application.



Features of Dependency Injection

- Injector provides a mechanism to create a service instance, and to create the instance, it needs a provider.
- Provider is a medium for creating a service.
- Providers can be registered along with the injectors.



Features of Dependency Injection

It is stimulated into the framework and can be reused.

DI creates a new instance of class and its required dependencies.

It empowers you to inject dependencies in different components across the application without the need to know how those dependencies are created.

DI is a big selling point for Angular2.

Dependency Injection and Service

Topic 2—Dependency Injection API

Importance of Dependency Injection API

Angular2 has its own dependency injection framework, which can also be used as a standalone module by other applications and frameworks.

DI in Angular2 consists of:

Injector

- It exposes the APIs to you to create instances of dependencies.

Provider

- It is a medium to create an instance of a dependency.
- It maps the selected token to a factory function that creates an object.

Dependency

- It is a technique where one object supplies the dependencies of another object.
- A Dependency is an object that can be used (a service).

Introduction to Multi Providers

A provider is an instruction that describes how an object is created for a particular token.

- In Angular2, the DI system offers the feature of Multi Providers.
- It provides consumers the capability to add custom functionality according to the need of the application.

Understanding Multi Providers

- You can provide multiple dependencies for a single token using Multi Providers.
- This code uses Multi Providers and manually creates an injector:

training@localhost:~

```
const SOME_TOKEN: OpaqueToken = new OpaqueToken("SomeToken");

var injector = ReflectiveInjector.resolveAndCreate([
      { provide: SOME_TOKEN, useValue: 'dependency one' , multi:
      true },
      { provide: SOME_TOKEN, useValue: 'dependency two' , multi:
      true },
]);

var dependencies = injector.get(SOME_TOKEN) ;

//dependencies == ['dependency one' , 'dependency two']
```

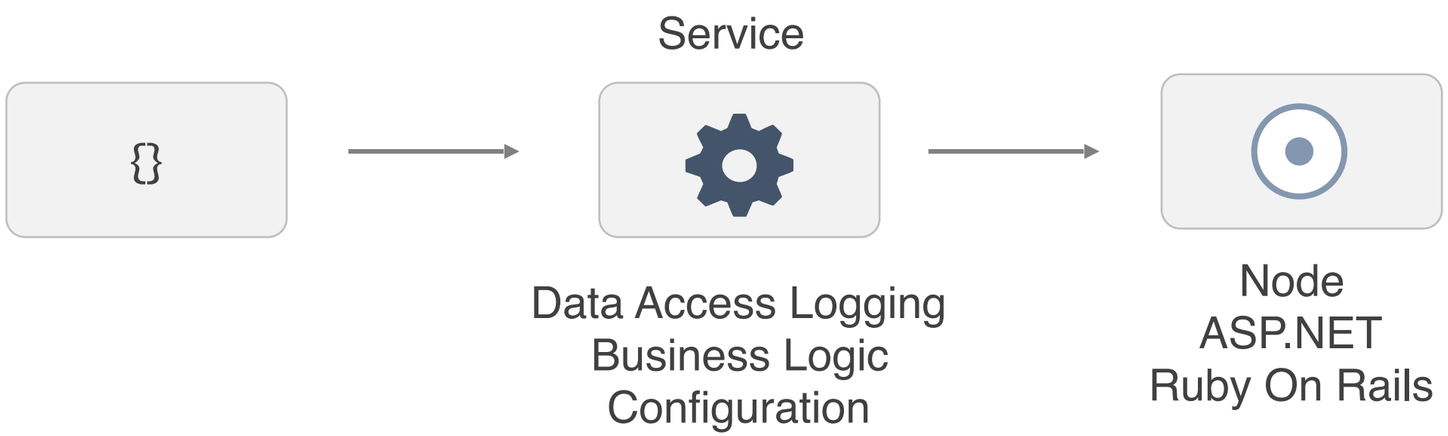
Note: In Angular 2, usually the platform creates the injectors and they are not created manually.

Dependency Injection and Service

Topic 3—Service Overview

Service

Service contains business logic, which has no relationship with view.



Introduction to Service

When multiple components need access to the same data, the best alternative to copying the same code again and again is to create a single data service that you can reuse and inject in the components that need it.

Use replicated storage levels MEMORY_ONLY_2, MEMORY_AND_DISK_2

Dependency Injection and Service

Topic 4—Creating a Service

Creation of Service

Import the Angular2 Injectable function and apply that function as an @Injectable() decorator.

training@localhost:~

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

@Injectable()
export class HeroService {
}
```

TypeScript looks at the @Injectable() decorator and emits metadata about the service. Metadata in Angular2 may need to inject other dependencies into this service.

Injecting a Service

Step 1: Import the Service you want to use so that you can reference it in the code.

training@localhost:~

```
import { HeroService } from './hero.service';
```

Injecting a Service

Step 2: Inject the HeroService.

Note: You can create a new instance of the HeroService with *new*:

training@localhost:~

```
heroService = new HeroService(); // don't do this
```

Instead, the one line of code (with new) is replaced with two lines:

- Add a constructor that also defines a private property.
- Add to the component provider's metadata.

training@localhost:~

```
constructor(private heroService: HeroService) { }
```

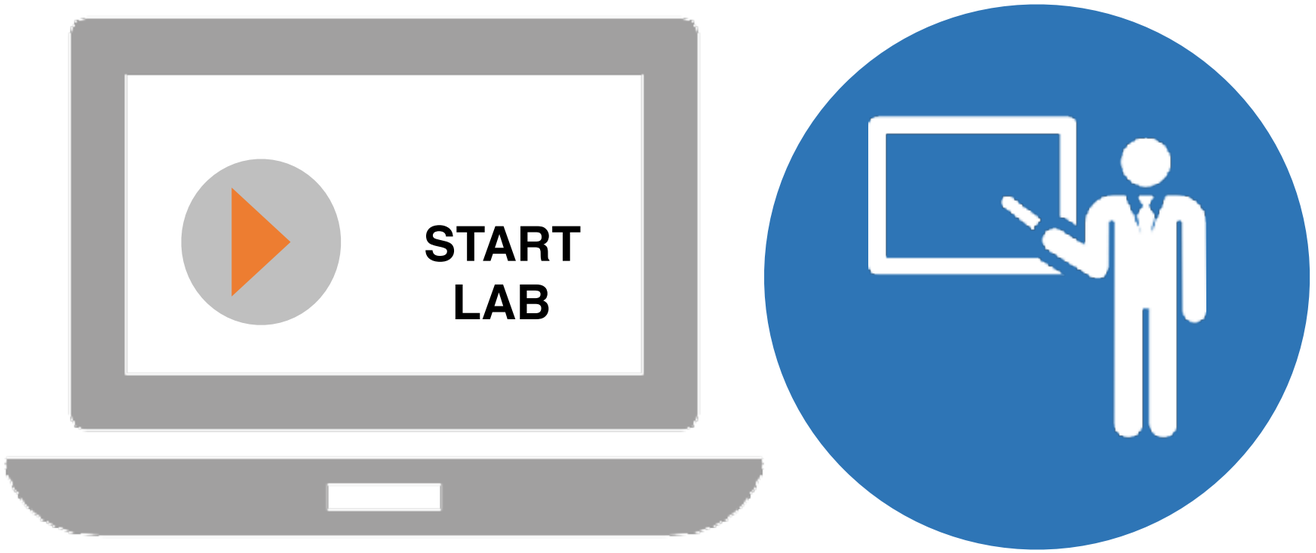
Injecting a Service

Step 3: Add the providers array property to the bottom of the component metadata in the @Component call to teach the injector the process to make a HeroService.

training@localhost:~

```
providers: [HeroService]
```

Lab—Demo



Key Takeaways

- Dependency Injection is a design pattern that passes an object as dependencies to different components across the application.
- Injection API consists of three things: injector, provider, and dependency.
- Service contains business logic, which has no relationship with view.
- To create a Service, use @Injector annotation.
- To use a Service, use constructor(private helloService: HelloService) { }.

Quiz

QUIZ 1	**Services are used to:**

a. Add behavior to DOM elements

b. Encapsulate any non-UI logic

c. Control navigation

d. None of the above

QUIZ 1	**Services are used to:**

a. Add behavior to DOM elements

b. Encapsulate any non-UI logic

c. Control navigation

d. None of the above

The correct answer is **b.**

Services are used to encapsulate any non-UI logic.

QUIZ 2	**What is the role of providers [ ] in module .ts file?**

a. It stores all the components

b. It stores all the directives

c. It stores all the services

d. All of the above

QUIZ 2	**What is the role of providers [ ] in module .ts file?**

a. It stores all the components

b. It stores all the directives

c. It stores all the services

d. All of the above

The correct answer is **c.**

Providers [ ] store all the services in module .ts file.

QUIZ 3	**A service can call another service.**

a. True

b. False

QUIZ 3	**A service can call another service.**

a. True

b. False

The correct answer is **a.**

Services can call another service.

Thank You