Using Services and Dependency Injection

Module Introduction

- The main objective of a **service** is to organize and share business logic, models, or data and functions with different components of an Angular application
- Should be robust and reusable
- Services are usually implemented through **dependency injection**

Creating a Logging Service

- CLI for generating services → **ng generate service <service-name>**
- Creating services manually
  - In an appropriate location, create **<service-name>.service.ts**
  - Create and export the service component class
  - Add the **@Injectable** decorator
    - Contains a **providedIn** field with a default value of **root**
- The service must be imported when used in other components
- You shouldn't manually create **service** instances

Injecting the Logging Service into Components

- **Dependency**
  - Something a given class will depend on
  - For instance, a component that use's a service's functions depends on that service
- **Dependency Injector**
  - Automatically injects an instance of the service into the target class
  - We must inform Angular of such an instance
- **How to Inject**
  - Import the service at the top of the file
  - In the recipient class's constructor, add an instance of the service
  - Example → **constructor(private simpleService: SimpleService) { }**
  - We must also provide the service so Angular knows how to create it

        - Add **providers** property to the component's **Component** decorator
        - Takes an array of service template names
- Angular now creates instances automatically

Creating a Data Service

- Services are excellent for storing and serving data
- Instead of storing more dynamic data in the **component** file, consider dumping it and their handler methods into the service file

Understanding the Hierarchical Injector

- Service instances created in one component are inherently shared with all child components
- **Hierarchy**
    - **AppModule** → The same instance of the service is available **application-wide**
    - **AppComponent** → The same instance of the service is available for **all components** (but **not for other services**)
    - **Any Other Component**
        - The same instance of the service is available for **the component and all its child components**
        - Services in child components **override** inherited service instances

How Many Instances of a Service Should There Be?

- Instantiating a service in a child component, despite inheriting said service from a parent component, yields different data → duh
- If you want to retain baseline service data, then don't list the service in the **providers** array

Injecting Services into Services

- Adding a service to the **providers** array of **AppModule** assures that all components share the same instance of said service unless overridden
- How to embed a service within another service

    - Add a **constructor** inside of the host service, including the instantiated hosted service
- To inject a service into something, that something must be accompanied by **metadata**
    - **Components** and **directives** have metadata because of their respective **@Component** and **@Directive** decorators
    - Services don't naturally include metadata
        - To appease Angular, inject the receiving service with **@Injectable()**
        - In modern versions of Angular, it's a standard to always include **@Injectable**

Using Services for Cross-Component Communication

- Not using services for cross-component communication entails **Input**ting, **Output**ting, and **emit**ting data with **property-** and **event-binding**
- Can exchange between components using a service
    - The service contains an **EventEmitter**
    - The outputting component invokes the service's **EventEmitter**'s **emit()** method
    - The inputting component **subscribes** to the service's **EventEmitter**, fetching the data via a response
        - This occurs in the constructor's body