Dependency Injection

- Services, Werte und Funktionen können injiziert werden
- Benötigt: Bereitstellung innerhalb eines Containers (Injector)
  - Bereitstellung durch Anhänge in providers-Liste
    - Innerhalb von Metadateninformationen für
      - Module
      - Komponenten

Service Wiederholung

Was, Wie, Wo?

1

ModulA

- Register (declarations)
    - KomponenteA

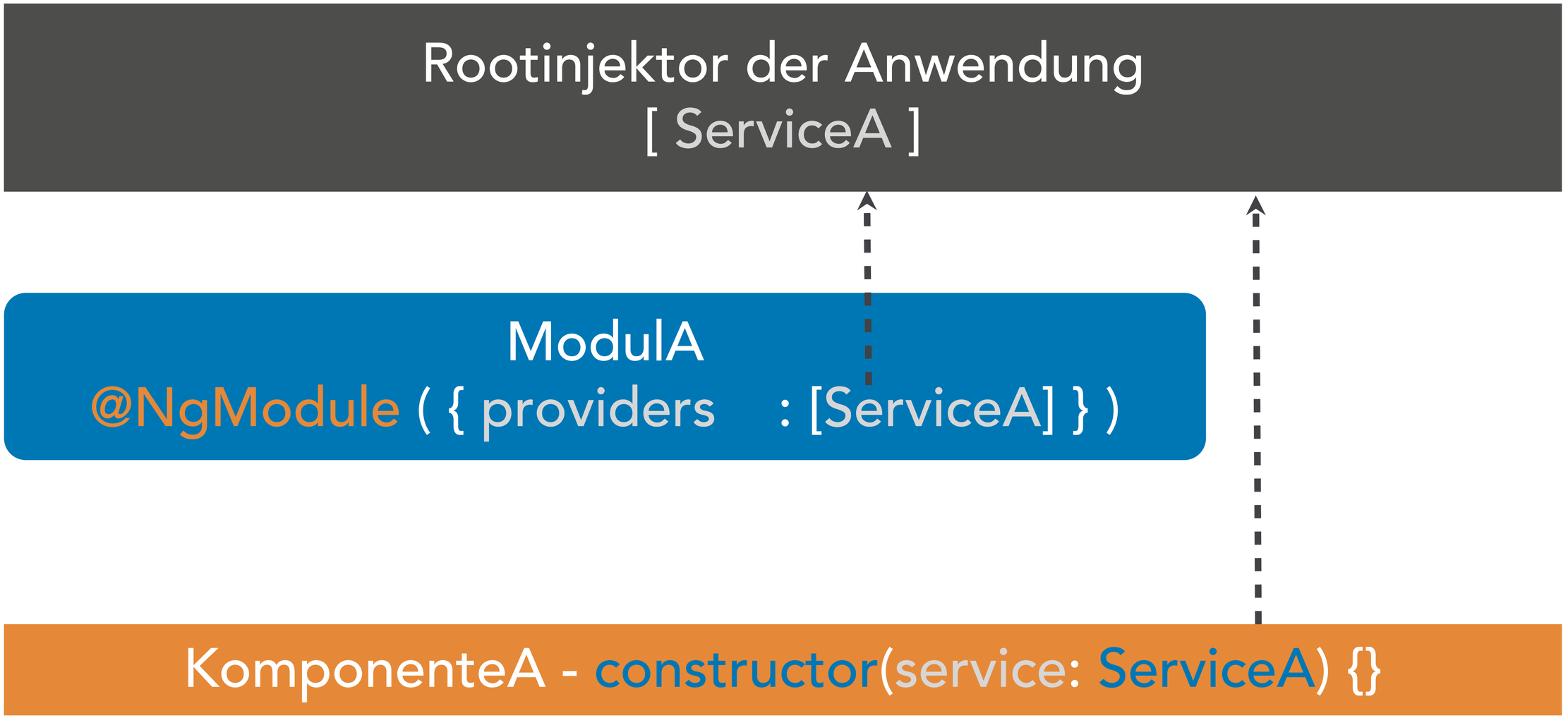
- Bereitstellen (providers)
    - ServiceA

2

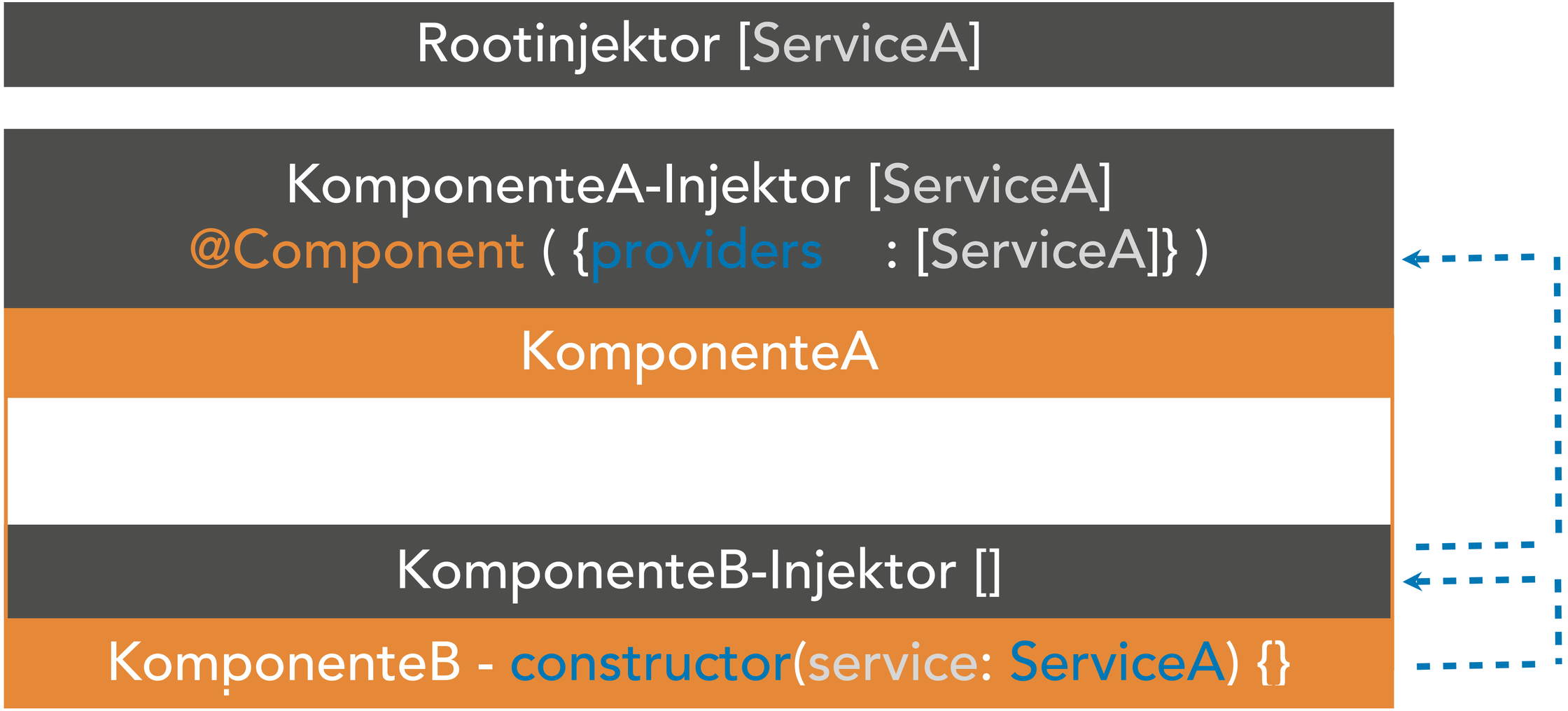
KomponenteA

```
constructor(
        service: ServiceA
) {
```

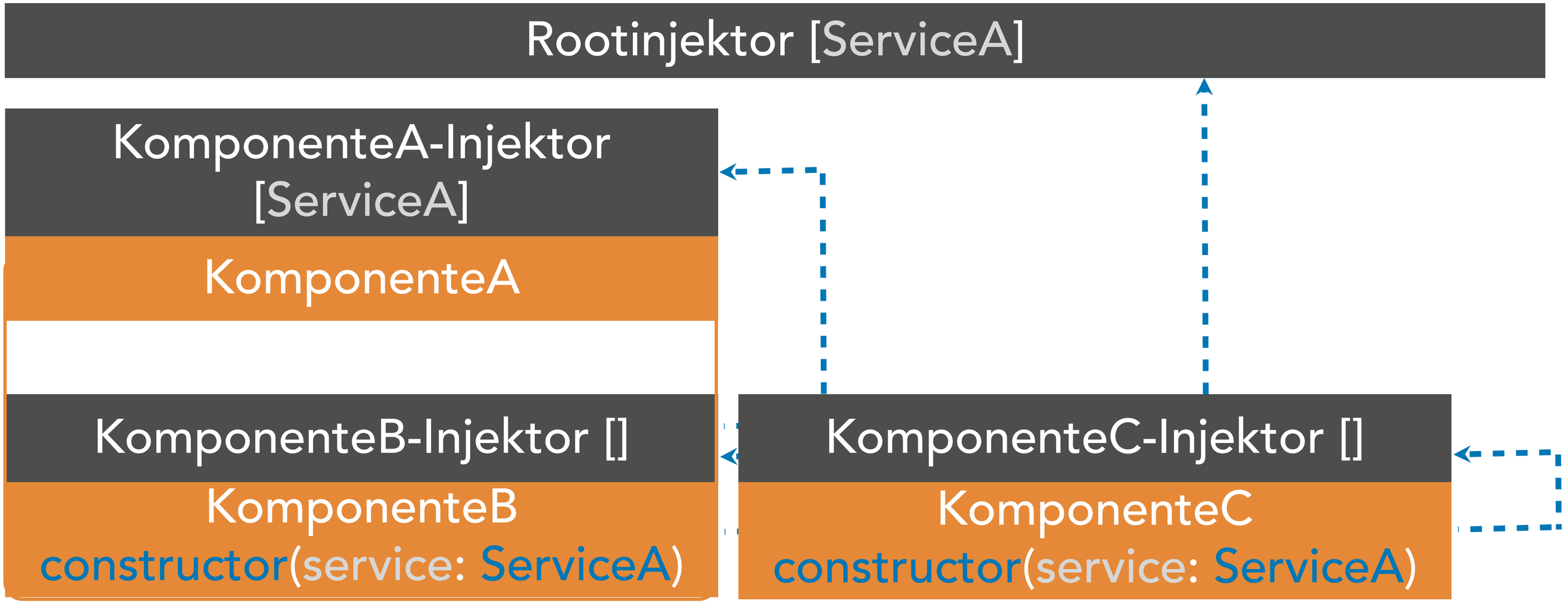
Dependency Injection



Dependency Injection



Dependency Injection



- *Erstelle ein users-Modul*
- *Provide einen user-Service darin*
- *Nutzen den Service in der App Komponente*

Übung 01

- *Erstelle ein user-Komponente*
- *Deklarier die Komponente im users-Modul*
- *Exportiere die Komponente im users-Modul*
- *Beweise, dass der user-Service ein Singleton ist.*

Übung 02

- *Erstelle ein user-list-Komponente*
- *Deklarier die Komponente im users-Modul*
- *Exportiere die Komponente im users-Modul*
- *Provide den user-Service in der user-Komponente*
- *Nutze <user-list> als Kind von <user>*
- *Beweise, dass es zwei user-Service Instanzen gibt*

Übung 03

Provide von Werten im Injector

- Nutzung von StaticProvider Typen statt Klassen
  - ValueProvider
  - ClassProvider
  - ExistingProvider
  - FactoryProvider

ValueProvider

- Werte im Injector registrieren
  - provide: any
    - Referenz zum injizieren
  - useValue: any –
    - Wert
  - multi?: boolean
    - Nutzung als Liste

Injizierten-Wert nutzen

- Werte die im Injector bereitgestellt wurden lassen sich Injizieren
  - @Inject Decorator
    - Referenz
    - Token

- *Nutze den ValueProvider*
- *Beweise, dass ohne multi die Werte überschrieben werden*
- *Zeige, dass mit multi eine Liste von Werten genutzt werden*

Übung 04

ClassProvider

- Klassen im Injector registrieren
  - Wie ValueProvider
  - useClass: Type<any> – statt ~~useValue~~
    - Klasse
      - sollte für aot im ES6-Modul exportiert sein

ExistingProvider

- Existierende Werte nutzen erneut registrieren
  - Wie ValueProvider
  - useExisting: any – statt ~~useValue~~
    - Referenz zu einem bereits registrierten Objekt

FactoryProvider

- FactoryMethode zum registrieren im Injector
- Wie ValueProvider
  - useFactory: Function – statt ~~useValue~~
    - Factory-Methode
  - deps: [any]
    - Liste von Abh.

DI-Decoratoren

- @Injectable – Zeichnet Service-Klassen aus, damit diese wiederum die DI im Konstruktor nutzen können
- @Inject - injiziert anhand eines Tokens
- @Optional – wir vor @Inject verwendet, ermöglichen optionale Injizierung
- @Self, @Host, @SkipSelf – wird genutzt, um das Injector-Bubling zu kontrollieren

InjectionToken

- Erzeugt Referenz-Token zu einer DI
  - Generische Type verweist auf Werte-Typ der DI

- *Erzeuge ein home-Modul und –Komponente*
- *Erzeuge eine home und user Route*
- *Beweise, dass UserService in beiden Routen erreichbar ist*

Übung 05

- *Refactore das Sample auf LazyLoaded Modules*
- *Erkunde, wo der UserService registriert werden muss.*

Übung 06