ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Dependency Injection Deep Dive

Alex Thalhammer

Content

- Classic Providers
- Tree-shakable Providers
- Scopes
- Constants as Tokens
- Services and Lazy Loading
- Multi Providers

Classic Providers

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Service

```
@Injectable()
export class FlightService {
    […]
}
```

Classic Providers

```
@NgModule({
    imports: [
        BrowserModule, HttpModule, FormsModule
    ],
    declarations: [
        AppComponent, FlightSearchComponent
    ],
    providers: [
        FlightService
    ],
    bootstrap: [
        AppComponent
    ]
})
export class AppModule {}
```

Standalone Components

```
bootstrapApplication(AppComponent, {
  providers: [
     FlightService
  ]
}
```

Injecting Service into Consumer

```
@Component({
    selector: 'flight-search',
    templateUrl: 'flight-search.html'
})
export class FlightSearchComponent {
    from = '';
    to = '';
    flight: Flight[] = [];

    constructor(private flightService: FlightService) { … }

    flightSearch(): void { [...] }
    selectFlight(flight): void { [...] }
}
```

Injecting Service into Consumer – NEW in V14

```
@Component({
    selector: 'flight-search',
    templateUrl: 'flight-search.html'
})
export class FlightSearchComponent {
    from = '';
    to = '';
    flight: Flight[] = [];

    private flightService = inject(FlightService);

    flightSearch() { [...] }
    selectFlight(flight) { [...] }
}
```

Abstraction

```
export abstract class FlightService {
    abstract find(from: string, to: string): Observable<Flight[]>;
}
```

```
@Injectable()
export class DefaultFlightService implements FlightService {
    find(from: string, to: string): Observable<Flight[]> { … }

}
```

"Forward" with useClass

```
@NgModule({
    imports: [
        BrowserModule, HttpModule, FormsModule
    ],
    declarations: [
        AppComponent, FlightSearchComponent
    ],
    providers: [
        { provide: FlightService, useClass: DefaultFlightService }
    ],
    bootstrap: [
        AppComponent
    ]
})
export class AppModule {}
```

"Forward" with useClass

```
@NgModule({
    imports: [
        BrowserModule, HttpModule, FormsModule
    ],
    declarations: [
        AppComponent, FlightSearchComponent
    ],
    providers: [
        { provide: FlightService, useClass: DefaultFlightService }
    ],
    bootstrap: [
        AppComponent
    ]
})
export class AppModule {}
```

"Forward" with useClass

```
@NgModule({
    imports: [
        BrowserModule, HttpModule, FormsModule
    ],
    declarations: [
        AppComponent, FlightSearchComponent
    ],
    providers: [
        { provide: FlightService, useClass: DefaultFlightService }
    ],
    bootstrap: [
        AppComponent
    ]
})
export class AppModule {}
```

Token

Service

Tokens

- Interfaces **cannot** be used as tokens
- However, you can use abstract classes instead

useFactory

```
const DEBUG = !environment.production;
[...]

providers: [{
    provide: FlightService,
    useFactory: (http: HttpClient) => {
        if (DEBUG) {
            return new DummyFlightService();
        } else {
            return new DefaultFlightService(http);
        }
     },
     […]
}]
```

useFactory

```
const DEBUG = true;
[...]

providers: [{
    provide: FlightService,
    useFactory: (http: HttpClient) => {
        if (DEBUG) {
            return new DummyFlightService();
        } else {
            return new DefaultFlightService(http);
        }
     },
     deps: [HttpClient]
}]
```

Types of Providers

useClass

useValue

useFactory

useExisting

Tree-Shakable Provider

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Tree-Shakable Provider

App scope

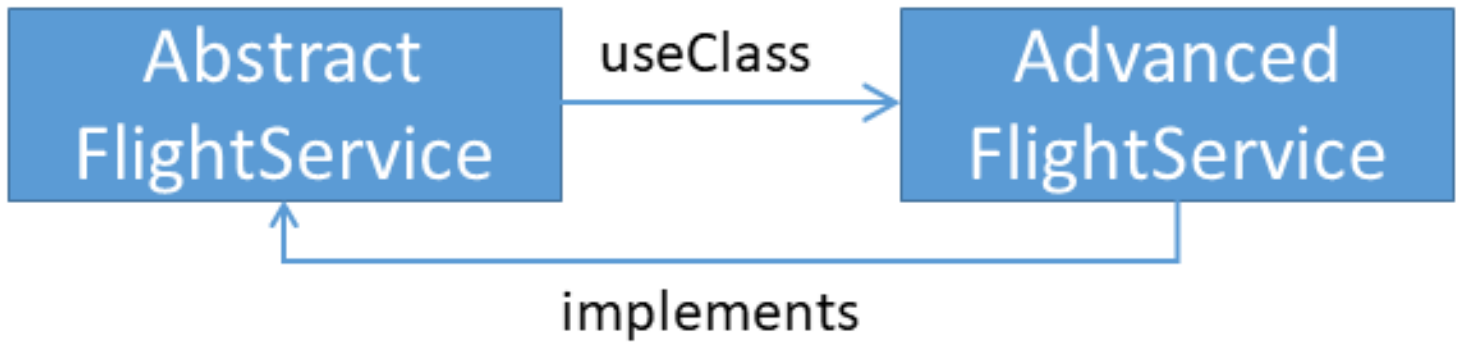
```
@Injectable({ providedIn: 'root' })
export class FlightService {
    […]
}
```

useClass

```
@Injectable({
    providedIn: 'root',
    useClass: DefaultFlightService
})
export abstract class FlightService {
    abstract find(from: string, to: string);
}
```

```
@Injectable()
export class DefaultFlightService implements FlightService {
    find(from: string, to: string) { … }
}
```

implements vs. *extends*



useFactory

```
@Injectable({
    providedIn: 'root',
    useFactory: (http: HttpClient) => {
        if (DEBUG) {
            return new DummyFlightService();
        } else {
            return new DefaultFlightService(http);
        }
    },
    deps: [HttpClient]
})
export abstract class FlightService {
    abstract find(from: string, to: string): Observable<Flight[]>;
}
```

DEMO

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Scopes

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

"Locale Service": Own Scope

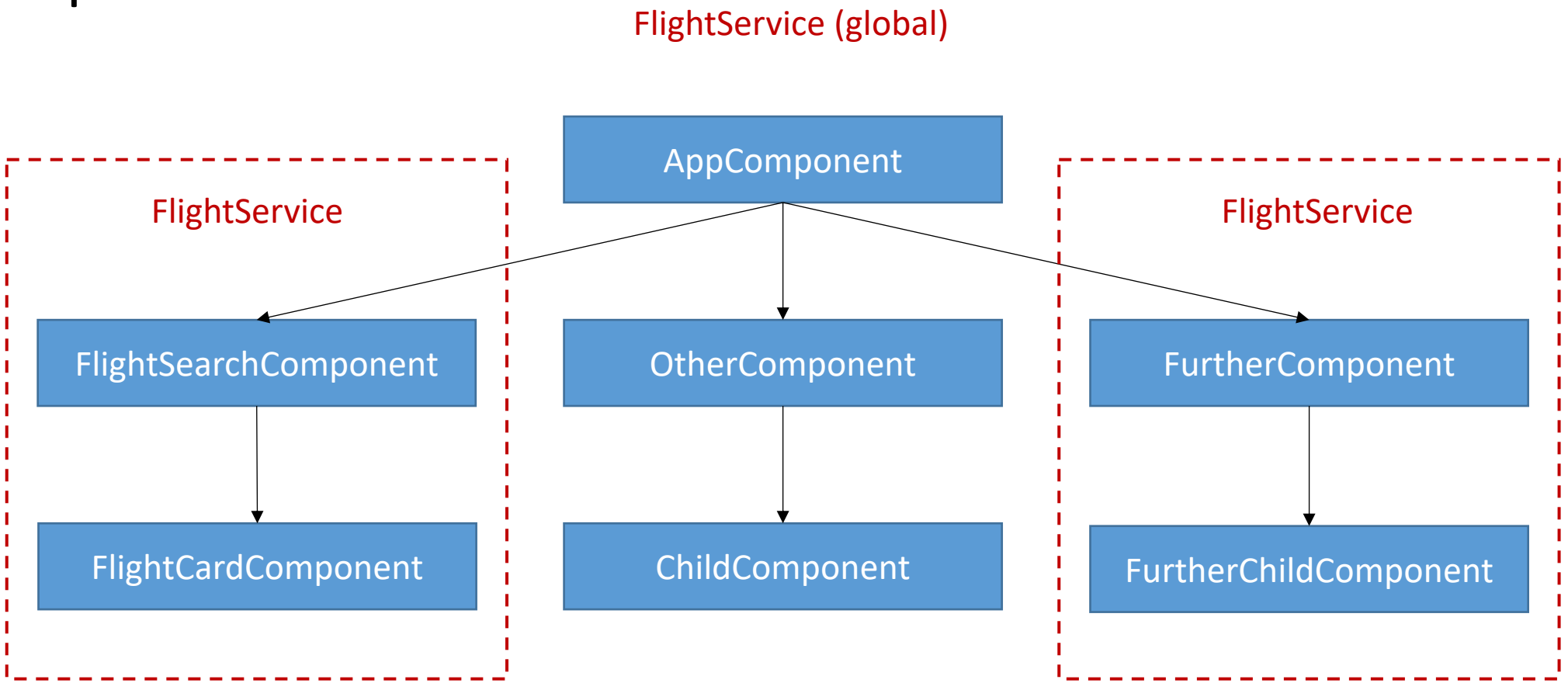
```
@Component({
    selector: 'app-flight-search',
    templateUrl: './flight-search.html',
    providers: [
       { provide: FlightService, useClass: DefaultFlightService}
    ]
})
export class FlightSearchComponent {

    […]

}
```

Can be used in current component and below!

Scopes


FlightService (global)
AppComponent
FlightService
FlightSearchComponent
FlightCardComponent
OtherComponent
ChildComponent
FlightService
FurtherComponent
FurtherChildComponent

DEMO

Multi Providers

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Multi

```
[...]
providers: [
      { provide: FlightService, useClass: LufthansaFlightService, multi: true },
      { provide: FlightService, useClass: AustrianFlightService, multi: true }
]
[...]
```

```
export class AppComponent {
   constructor(
      @Inject(FlightService) private flightServices: FlightService[]) {
   }
   [...]
}
```

Multi and Constants

```
[...]
providers: [
      { provide: FLIGHT_SERVICES, useClass: LufthansaFlightService, multi: true },
      { provide: FLIGHT_SERVICES, useValue: AustrianFlightService, multi: true }
]
[...]
```

```
export class AppComponent {
   constructor(
      @Inject(FLIGHT_SERVICES) private flightServices: FlightService[]) {
   }
   [...]
}
```

Summary

- forRoot vs. in component
- Classic providers vs. tree-shakable providers
- Factories
- Multi Providers