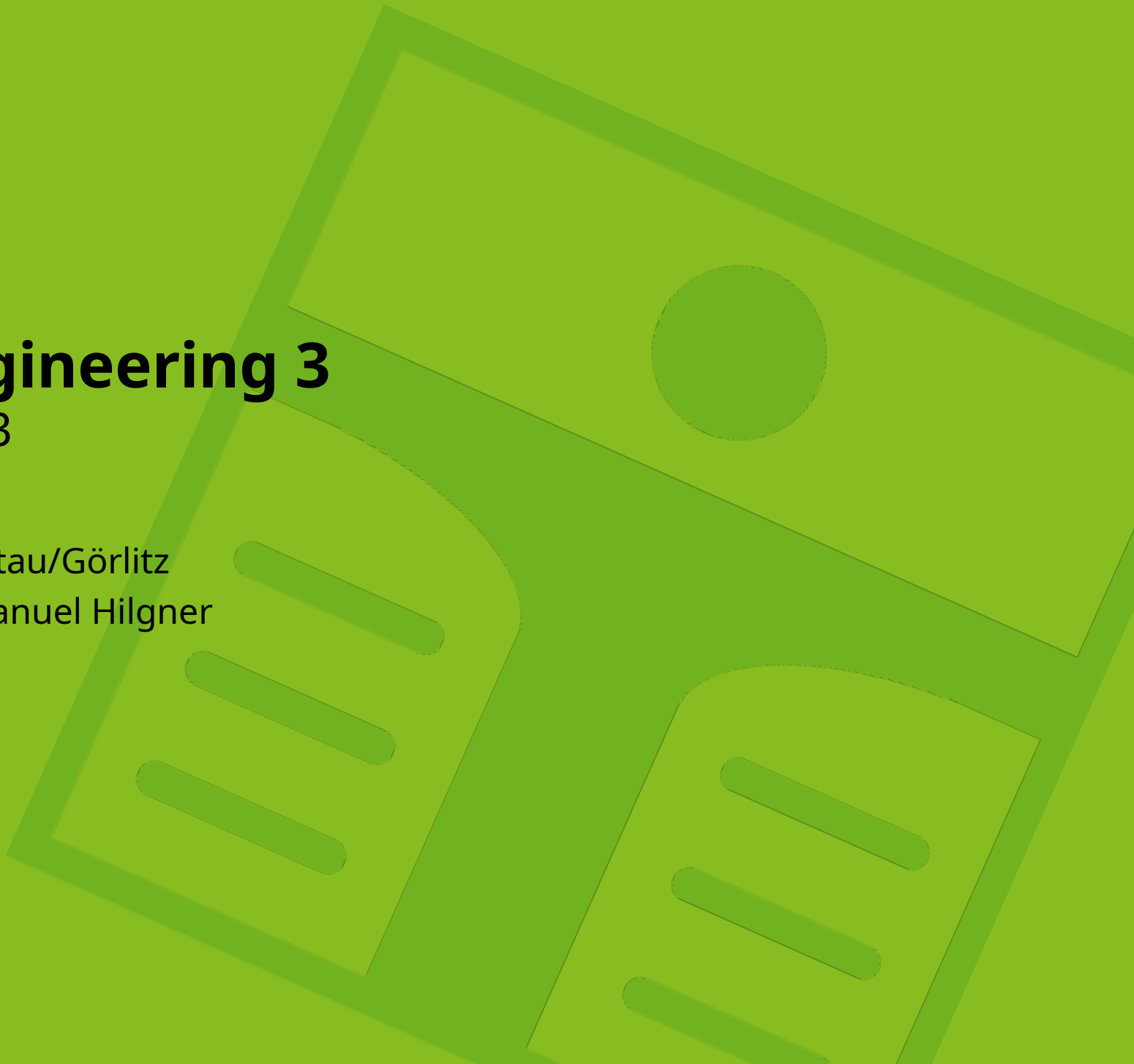
Web Engineering 3

Vorlesung 3

Hochschule Zittau/Görlitz
Christopher-Manuel Hilgner



Agenda

Wie & Warum funktioniert Spring eigentlich?

- Inversion of Control
- Dependency Injection
- Spring Beans
- Spring Data
- HTTP Methoden

Inversion of Conrol

- Kernpunkt vieler Frameworks, die das Hinzufügen und Erweitern von Funktionalitäten erlauben
- Framework erhält Kontrolle über alles, was der Nutzer geschrieben hat
- Folgendes Prinzip aus Hollywood: **“Don’t call us, we’ll call you”**

Inversion of Control

- Framework ruft vom Nutzer geschriebene Funktionen auf
- Framework stellt Hauptprogramm dar
- Koodiniert Aktivitäten der Anwendung
- Erweiterbares Skelett für eine Anwendung
- Nutzer können generische Algorithmen auf spezifische Anwendungsfälle zuschneiden

Dependency Injection

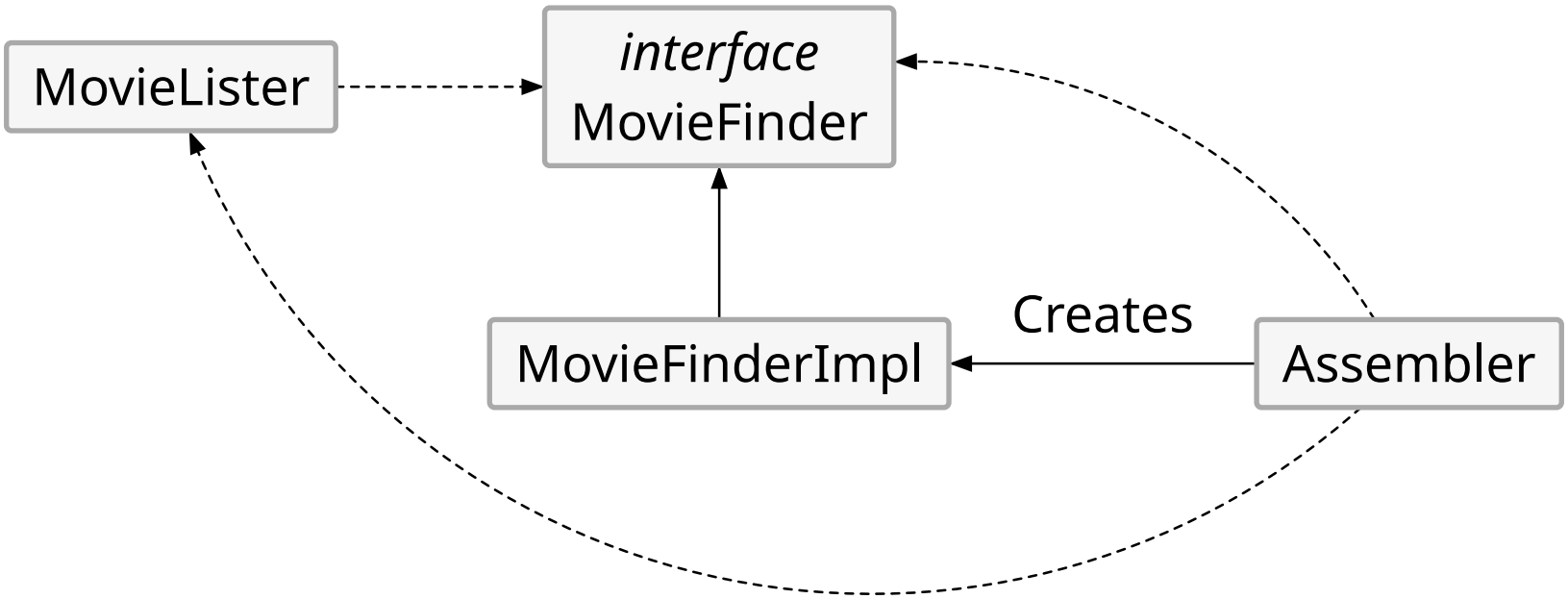
- **Ziel**: Entkoppelung von Abhängigkeiten
- Entkoppelung macht Testen einfacher und erhöht Lesbarkeit
- Klassen müssen nur noch ihre Abhängigkeiten definieren
- Abhängigkeiten werden durch Container bereitgestellt

Dependency Injection

- Assembler Objekte bevölkern Felder in Klassen nach definierten Anforderungen
- Kann Implementation von einem Interface sein
- Drei Möglichkeiten der Dependency Injection

1. Contructor Injection bzw. Type 3 IoC
2. Setter Injection bzw. Type 2 IoC
3. Interface Injection bzw. Type 1 IoC

Beispiel von Dependency Injection



- `MovieLister` Klasse benötigt Implementation von `MovieFinder`
- Assembler stellt Implementation bereit und erfüllt Abhängigkeit

IoC Container

- Der IoC Container wird durch `ApplicationContext` representiert
- Instantiierung, Konfiguration und Assembling von Beans
- Intruktionen dafür werden durch Konfigurations-Metadaten übergeben
- Wege der Konfiguration:
  - Annotations
  - Konfigurations-Klasse mit Factory Methoden
  - XML-Dateien
  - Groovy Scripts

IoC Container

- Manuelle Erstellung ist nicht von Nöten
- Spring kombiniert die erstellten Klassen mit Konfigurations-Metadaten
- Nach der Initialisierung von `ApplicationContext` steht ein ausführbares, konfiguriertes System bereit

Dependency Injection in Spring

- Abhängigkeiten werden in Contructor, Factory Methoden oder Properties definiert
- In Spring: Contructor oder Setter Methoden
- Container übergibt beim Erstellen einer Bean alle Abhängigkeiten
- Bean hat keine Kontrolle über Erstellung oder Ort ihrer Abhängigkeiten

Contructor oder Setter DI

Richtlinie 1

Der Contructor sollte verpflichtende Abhängigkeiten enthalten.

Setter Methoden eignen sich gut für optionale Abhängigkeiten. `@Autowired` kann bei Settern genutzt werden, damit die Property eine verpflichtende Abhängigkeit wird. Der Constructor sollte da aber bevorzugt werden.

Contructor Injection

- Container ruft Contructor auf mit so vielen Argumenten wie Abhängigkeiten
- Jedes Argument ist eine Abhängigkeit

```kotlin
class ExampleClass(private val dependency: Dependency)
{

}
```

Setter Injection

- Leerer Contructor wird aufgerufen
- Container ruft Setter Methode in erstellten Beans auf

```kotlin
class ExampleClass {
  lateinit var dependency: Dependency
}
```


me and the boys at 2am
looking for
BEANS

Beans

Beans Definition 2

Jedes Objekt, welches Teil der Anwendung ist und von dem Spring IoC Container verwaltet wird, ist eine Bean. Eine Bean kann instantiated, assembled oder anderweitig von dem Spring IoC container gemanaged werden.

Bean Annotationen

- `@Component`: Eine generelle Angabe, die eine Klasse als Spring Bean markiert
- `@Service`: Eine Klasse, die einen Service darstellt
- `@Repository`: Eine Klasse, die ein Repository darstellt, welches mit der Persistence-Layer interagiert
- `@Controller`: Eine Klasse, die einen Controller, im Spring Model-Veiew-Controller darstellt

Bean Scopes

- Singleton
- Prototype
- Request
- Session
- Application
- WebSocket

Singleton Pattern

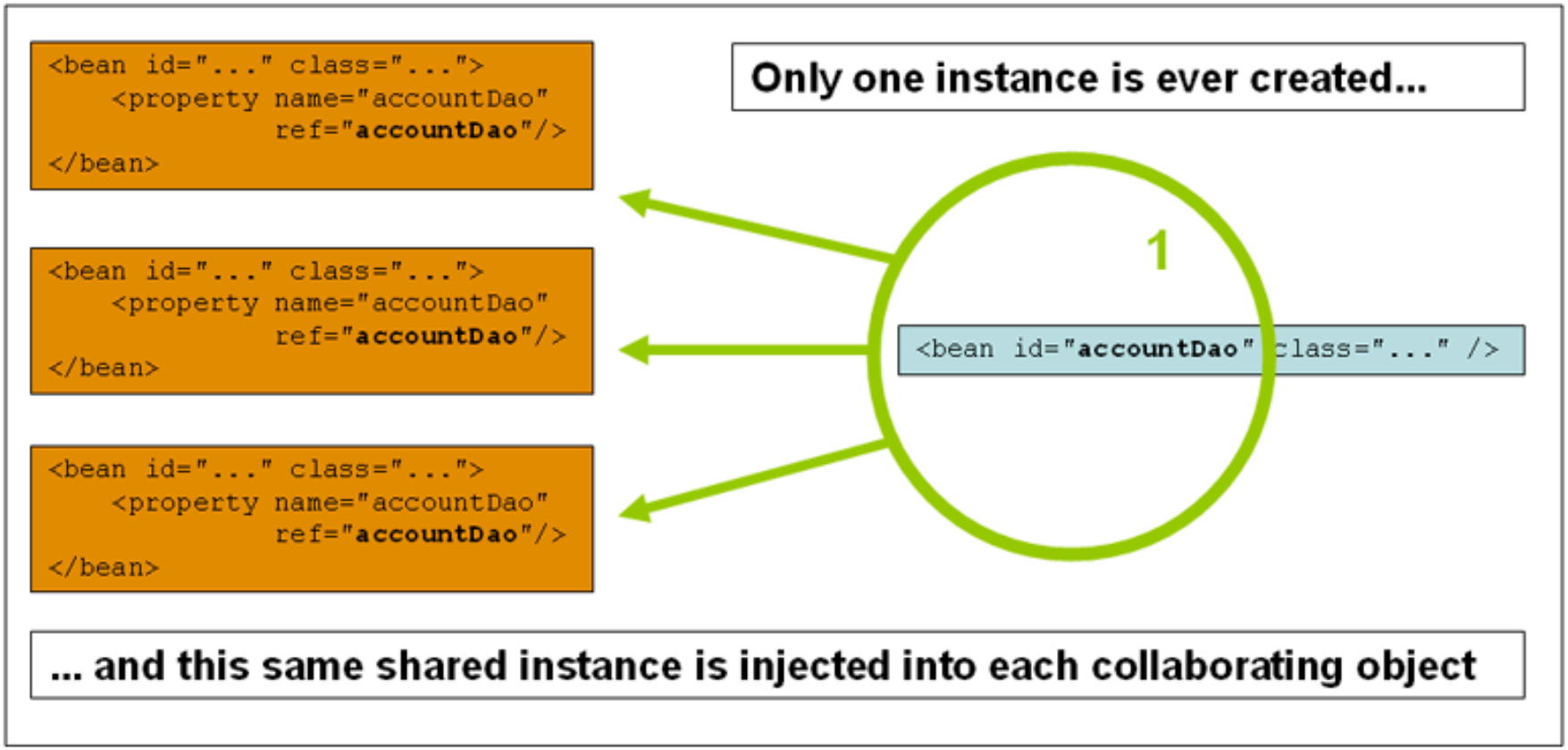
Bean Scopes - Singleton

- Eine Instanz einer Bean wird in gesamter Anwendung genutzt
- Wird in Chache aus Singleton-Beans gespeichert
- Jede Anfrage oder Referent auf diese Bean wird auf Cache verwiesen
- Singleton ist Standard für Beans

XML

```
1   <bean
2     id="accountService"
3     class="com.something.DefaultAccountService"
4   />
```

Bean Scopes - Singleton



Bean Scopes - Singleton

Einsatz von Singleton Beans Richtlinie 3

Singleton Beans sollten für stateless Beans eingesetzt werden.

Prototype Pattern

Bean Scopes - Prototype

- Eine neue Instanz wird bei jeder Anfrage erstellt
- Anfrage kann durch Injection in eine andere Bean oder durch Funktionsaufruf geschehen

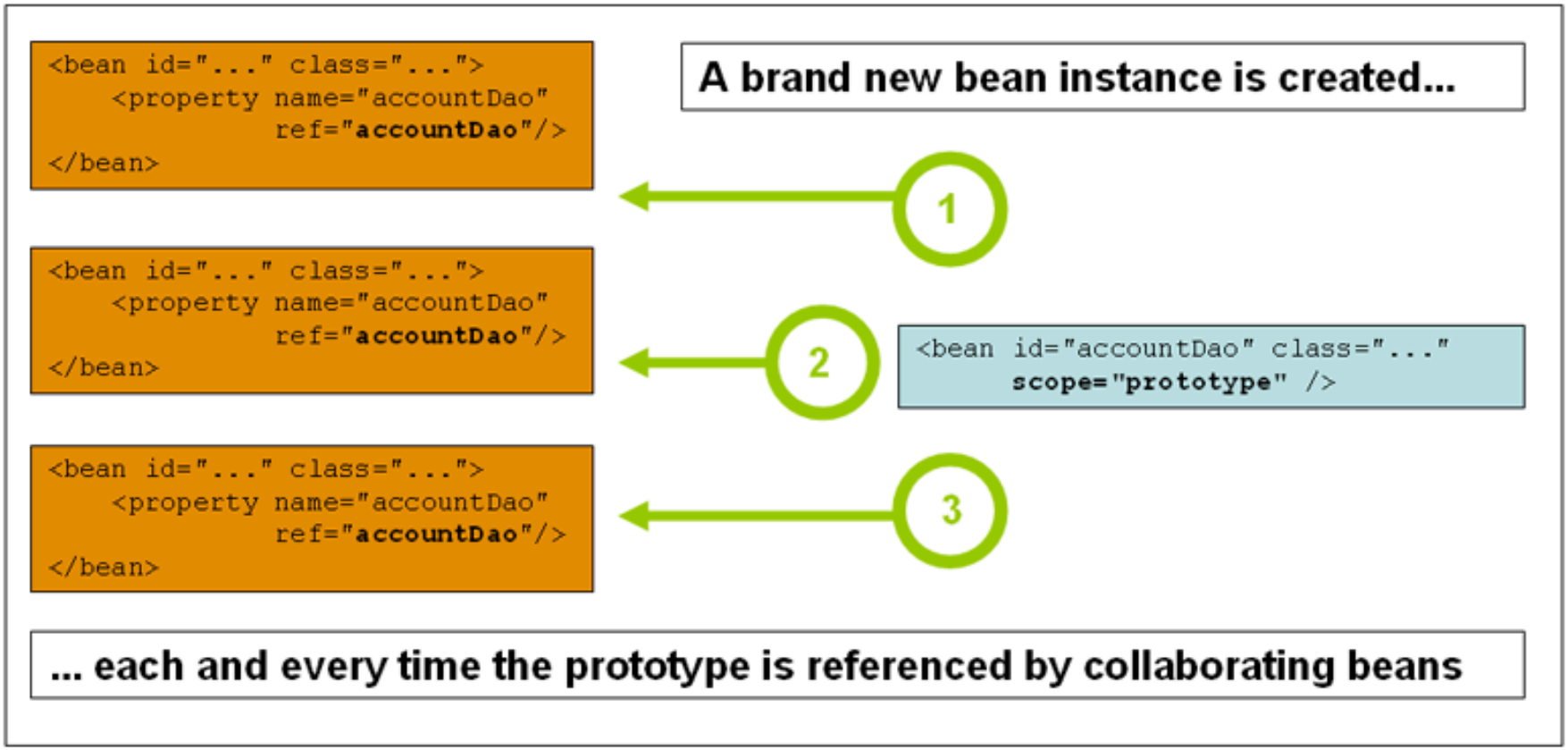
```kotlin
getBean()
```

```xml
<bean
  id="accountService"
  class="com.something.DefaultAccountService"
  scope="prototype"
/>
```

Bean Scopes - Prototype

- Spring verwaltet nicht kompletten Lebenszyklus
- Löschen muss durch Nutzer geschehen
- Selbst geschriebener Bean Post-Processor kann genutzt werden um Ressourcen freizugeben

Bean Scopes - Prototype



Bean Scopes - Request

- Eine einzelne Instanz für jede HTTP Anfrage
- Bean existiert nur so lange, wie die HTTP Anfrage beantwortet wird
- Andere Beans vom gleichen Typ sehen Änderungen nicht
- Sobald die Anfrage bearbeitet wurde, wird die Bean entfernt

XML

```xml
<bean
  id="loginAction"
  class="com.something.LoginAction"
  scope="request"
/>
```

Bean Scopes - Request

```kotlin
@RequestScope
@Component
class LoginAction {
  // ...
}
```

Bean Scopes - Session

- Eine einzelne Instanz für jede HTTP Session
- Bean wird auf Session gescoped
- State der Bean kann nur geändert werden, wenn die Session aktiv ist
- Andere Beans vom gleichen Typ sehen Änderungen nicht
- Beim Ende der Session wird die Bean entfernt

XML

```xml
<bean
  id="userPreferences"
  class="com.something.UserPreferences"
  scope="session"
/>
```

Bean Scopes - Session

```kotlin
@SessionScope
@Component
class UserPreferences {
  // ...
}
```

Bean Scopes - Application

- Eine Bean für die gesamte Web Anwendung
- Bean wird auf `ServletContext` gescoped
- Bean wird als Attribut von `ServletContext` gespeichert
- Unterschiede zu Singleton Beans:
  - Es existiert eine Bean pro `ServletContext`
  - Es wird exposed als Attribut von `ServletContext`

XML

```xml
<bean
  id="appPreferences"
  class="com.something.AppPreferences"
  scope="application"
/>
```

Bean Scopes - Application

```kotlin
@ApplicationScope
@Component
class AppPreferences {
  // ...
}
```


Mio
Mensa im Osten
Studentenwerk Dresden