JAVACREAM

Training
Consulting
Projectmanagement

Spring Grundlagen

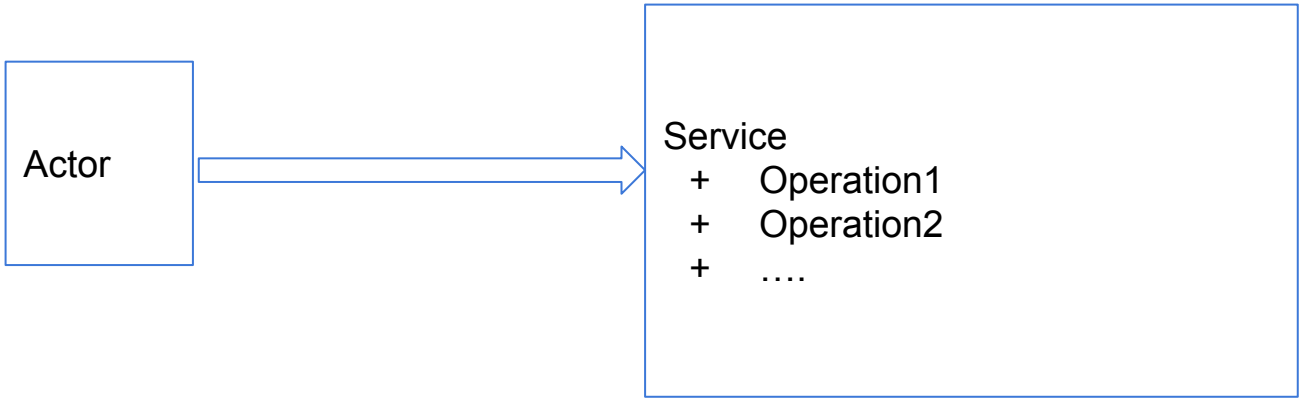
Kurze Vorstellung

JAVACREAM
Training
Consulting
Projectmanagement

- Name
- Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Aktuelle Problemstellung
- Konkrete individuelle Zielsetzung

Ausgangssituation

Fachanwendung



Bei Modellierung einer Fachanwendung keinerlei Bezug zu Spring vorhanden

Technisches Modell

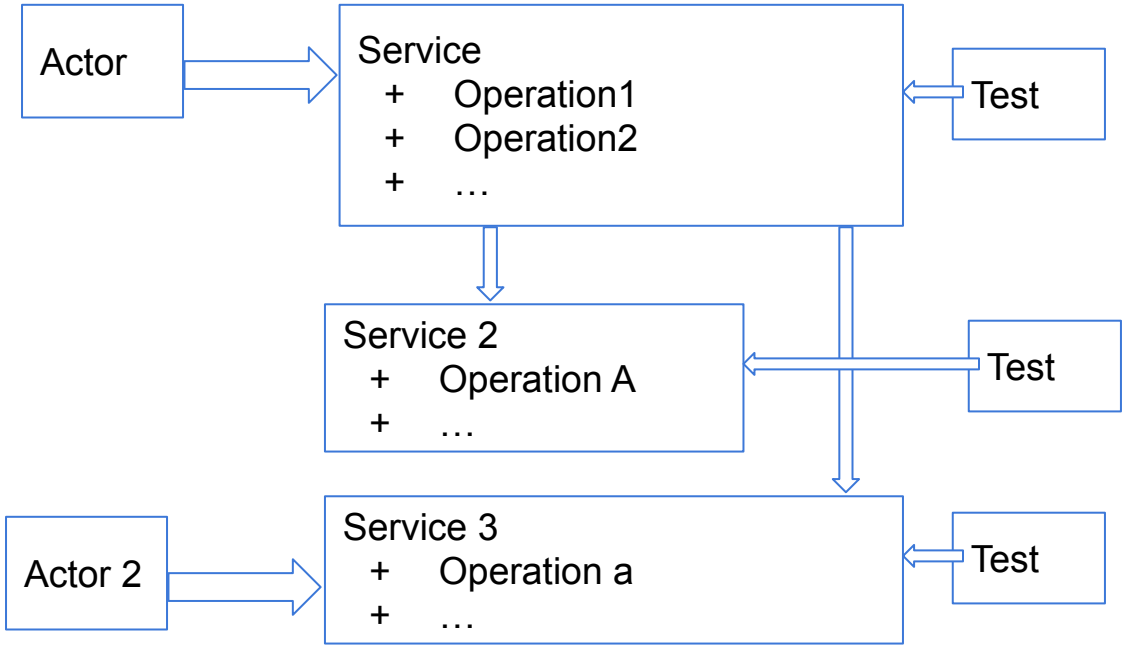
Anforderungen an das Modell

- \+ Wartbarkeit
- \+ Wiederverwendung
- \+ Testbarkeit

Umsetzung durch Modularisierung statt einer monolithischen Applikation

Bezug zu Spring ist indirekt

- \+ Bei Verwendung von Spring ist die Modularisierung eines technisches Modells sehr gut möglich

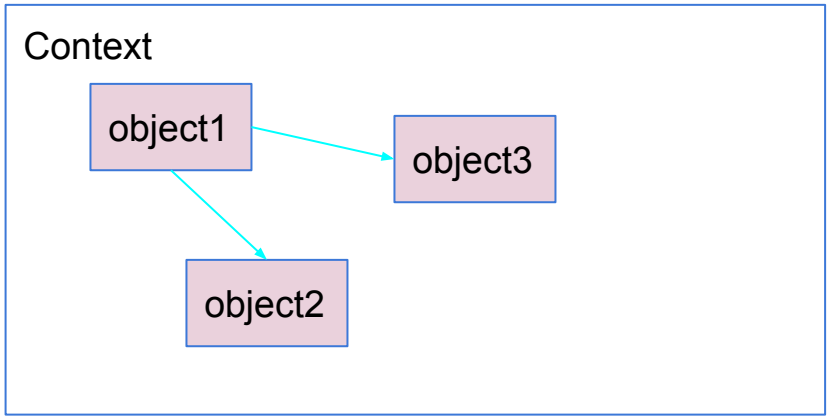


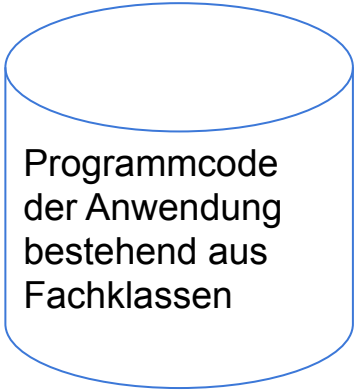
Context & Dependency Injection (CDI)

CDI baut aus den einzelnen Modulen das Objektgeflecht der Anwendung auf

Spring ist eine Umsetzung des Design Patterns Context & Dependency Injection

“Spring ist ein CDI-Framework”



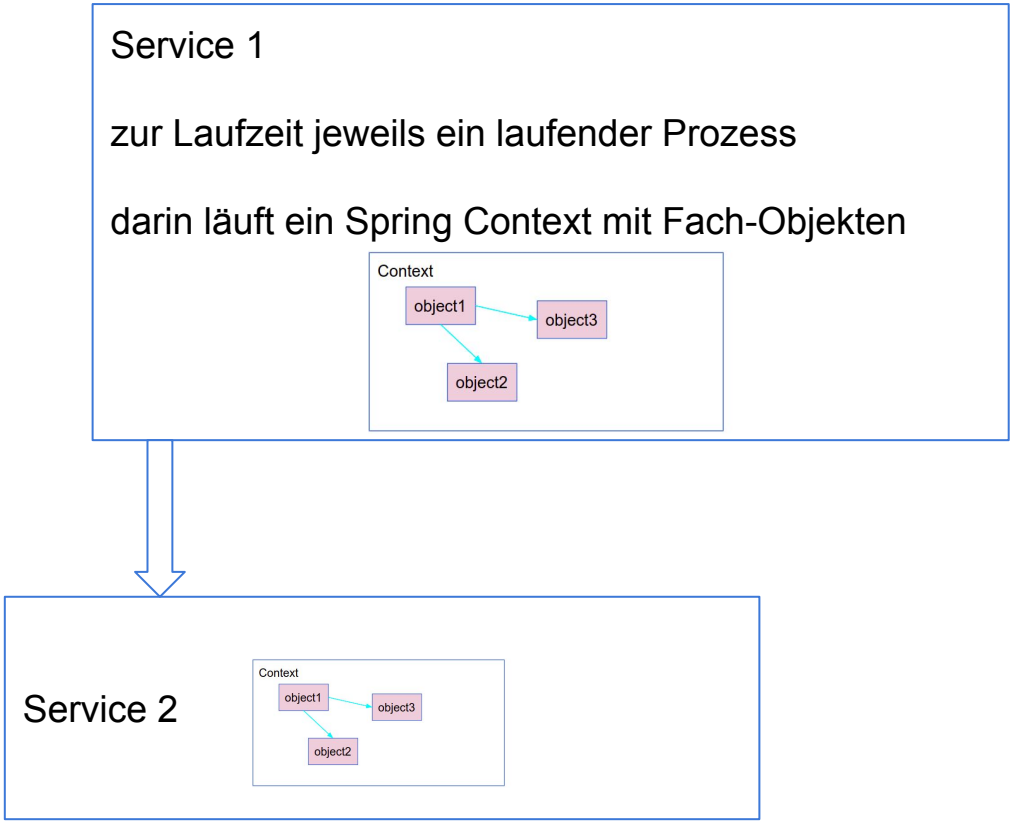


Aufgabe des Contexts

+ Identifikation der relevanten Fachklassen und Instanzierung von Fach-Objekten
+ Identifikation der Abhängigkeiten der Objekte und das Setzen der Abhängigkeit

Exkurs: Module versus Fachklassen

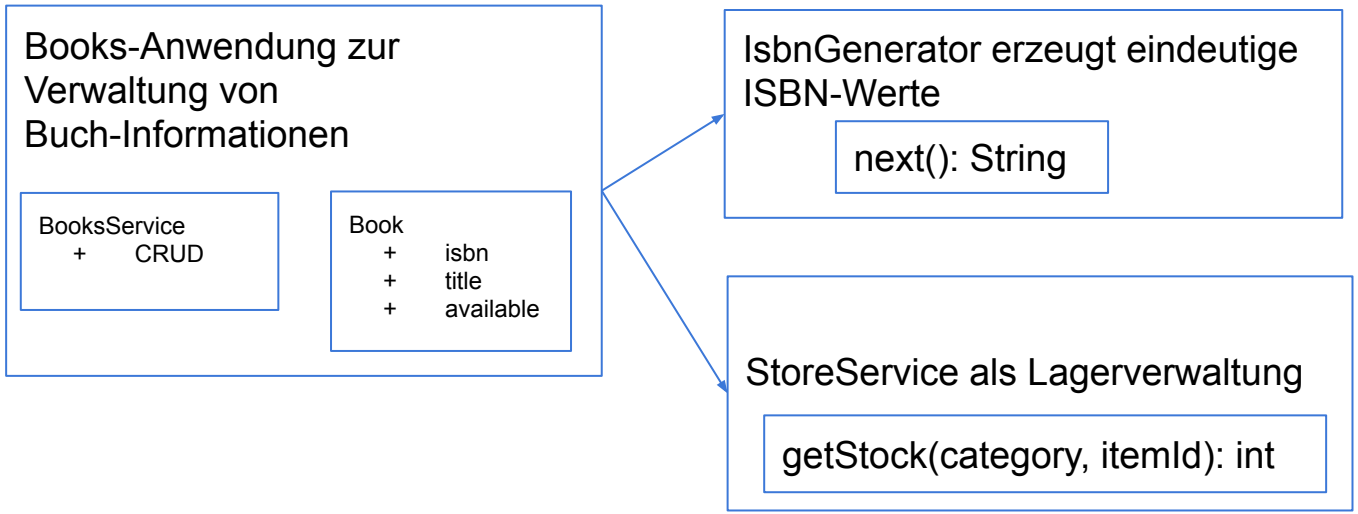
Service-oriented bzw. Microservices



Die Fachanwendung des Trainings

Modell

- Vollständig vorgegeben
- Fachlich einfach



Umsetzung

- Ist auch bereits vorhanden
  - Bisher
    - Die gesamte Datenhaltung In Memory
    - Zugriff ist nur für Actors im selben Prozess möglich
      - Actors = Test-Fälle
- Programmierung ist typisch für eine statisch typisierte Programmiersprache wie Java
  - Operationen sind in Schnittstellen definiert
  - Datenstrukturen sind simple Daten-Container (eigentlich structs oder records)
  - Operationen und Datencontainer definieren das API einer Fachanwendung
  - Zugehörige Implementierung ist eine Klasse, die die Schnittstelle implementiert

Umsetzung

- Bisher
  - Die Anwendung hat keinerlei Bezug zu Spring!
  - Die Anwendung selber ist jedoch CDI-konform!
    - Verifizierung: Relevante Fachklassen (MapBooksService, SimpleStoreService, RandomIsbnGenerator, CounterIsbnGenerator) werden im Rahmen der Anwendung NIEMALS mit new instanziert
    - Dependency ist ein Attribut vom Typ einer API-Schnittstelle + setter-Methode
      - Das ist das GoF-Pattern “Strategy”, das CDI-Pattern ist eine Meta-Pattern aus Strategy und Factory
  - Der Testfall übernimmt die Aufgaben des Contextes
    - new-Operatore
    - Aufruf der setter-Methoden

Spring First Contact

- Exkurs: “Spring” oder “Spring Boot”?
  - Spring = Spring Core ist das CDI-Framework
  - Spring Boot
    - Vereinfachter Build-Prozess
      - Dependency Management mit parent-pom und startern
    - Autoconfigure
      - “Convention over Configuration”
        - Welche Pakete sollen nach Spring-Informationen durchforstet werden?
        - Es wird automatisch eine Konfigurationsdatei namens application.properties eingeladen
  - Spring Core ist prinzipiell unabhängig von Spring Boot, aber es ist fast sinnlos, kein Spring Boot zu benutzen

Programmiermodell von Spring

- So nicht:
  - keine Namenskonventionen
  - benutzt keine Spring-Schnittstellen
    - relevant = “implements ContextAware”
- sondern ausgerichtet auf die Bereitstellung von Meta-Informationen
  - Externe XML-Konfiguration
  - **Java Annotations** (C#: Attributes)
    - @Component
      - oder @Service oder @Repository -> später
    - @Autowired
      - Referenzen auf Spring-relevante Objekte
    - @Value
      - Konfiguration auf einen Key, der in der application.xml eingetragen ist

Programmieren mit Spring

Spring Boot

- @SpringBootApplication
  - @SpringConfiguration
    - damit ist sie Spring-relevant
  - @EnableAutoconfiguration
    - z.B. laden der application.properties|yaml
  - @ComponentScan (“alles in diesem Paket und darunter”)
- @SpringBootTest
  - Scanne das gesamte Projekt nach einer @SpringConfiguration

@Component-Annotations

- Sind “Stereotypen”
  - Keine technische Unterschiede
  - Praktisch für die Dokumentation
- @Component
  - allgemein: “eine Spring-relevante Klasse”
- @Service
  - “Eine Klasse, die Service-Operationen anbietet”
- @Repository
  - “Eine Klasse, die CRUD-Operationen für eine Ressource anbietet”

Autowiring

- Beim Autowiring muss der Context exakt eine geeignete Spring Bean finden
  - Falls das nicht der Fall ist -> Fehler beim Hochfahren des Context
- @Autowired hat als Parameter nur “required = true|false”
  - kann damit auch auf null stehen
- Kandidaten zum Autowiring werden durch das Java Typsystem gefunden
  - Im Detail -> später
-

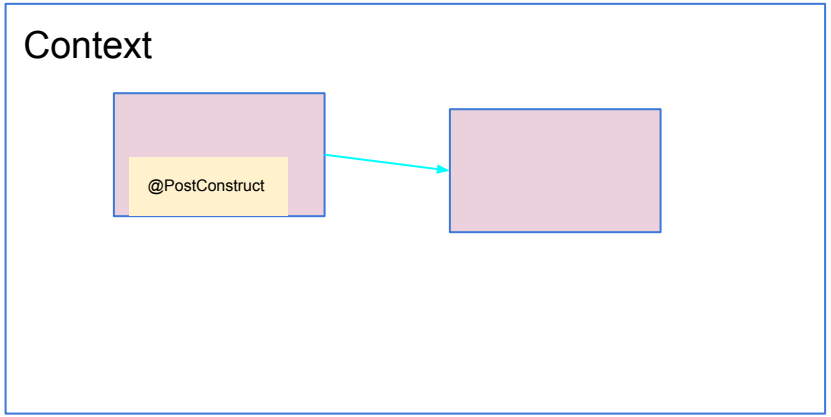
@Value

- Diese ist in der Lage, eine Konfigurationseinstellung zu injected
  - Spring Expression Language
    - ${}
    - Wert
      - Das zu verwendende Objekt
      - **object.property**
      - object.property.property
      - object.method()
- Nicht-vorhandene Konfigurationseinstellungen führen in Standard-Konfiguration zu einem Fehler

Weitere Features

- Erweiterung der Arbeitsweise des Context
  - Relevante Klassen können einen Lifecycle definieren
    - @PostConstruct public void <name>(){}
    - @PreDestroy public void <name>(){}
  - ToDo: Diskussion: Was ist der Unterschied zum Konstruktor?
- Eine Fachklasse kann auch einen parametrisierten Konstruktor aufweisen
  - Parameter werden automatisch als @Autowired aufgefasst
    - ToDo: MapBooksService mit Konstruktor-Parametern
    - ToDo: Diskussion Attribute-Injection versus setter-Injection versus Constructor-Injetion
      - Sind setter-Methoden notwendig?
  - Parameter können auch zur Value-Injection benutzt werden
    - ToDo: RandomIsbnGenerator mit Constructor mit prefix und countryCode

Lifecycle im Context



Aufgabe des Contexts

+ Identifikation der relevanten Fachklassen und Instanzierung von Fach-Objekten
  + Aufruf des Konstruktors
+ Identifikation der Abhängigkeiten der Objekte und das Setzen der Abhängigkeit
+ Aufrufen der @PostConstruct-Methoden

Autowiring im Detail

- Bisher
  - Der Context benutzt das Java Typsystem
    - Berücksichtigt wird hier auch Vererbung / Implementierung von Schnittstellen
    - RandomIsbnGenerator ist ein IsbnGenerator und ein Object
- Neu
  - Der Name des Injection Points (Attribut-Name, Constructor-Parameter-Name, setter-method ohne set + klein) wird ebenfalls benutzt
  - -> nächste Seite, @Resource
  - @Primary
    - bei Mehrdeutigkeiten wird @Primary benutzt
  - Qualifiers
    - Identifizierbar über eine Zeichenkette

ToDo

- Nachvollziehen der Präsentation zum Thema “resolve”
- Überlegen Sie, ob das aktuelle Paket der Strategy-Annotations wirklich gut ist?
  - Bessere Möglichkeit?

Scopes

- Wenn “jemand” eine Dependency benötigt, kann der Context entweder
  - eine bereits vorhanden Instanz benutzen oder
    - @Scope(”singleton”)
  - pro Injection Point eine neue Instanz erzeugen
    - @Scope(”prototype”)
- Standard-Scope ist “singleton”

@Autowiring versus @Resource

- Funktionsweise eines CDI-Frameworks
  - statisch autowiring
  - statisch deterministisch
  - ~~dynamisch autowiring~~
  - ~~dynamisch deterministisch~~
- Autowiring
  - Die Dependency wird durch einen Context-Algorithmus bestimmt
- Deterministisch
  - Der Entwickler legt die Dependency hart fest

Dependency Injection wird beim Hochfahren des Context identifiziert und gesetzt

Spring ist nicht dynamisch!