

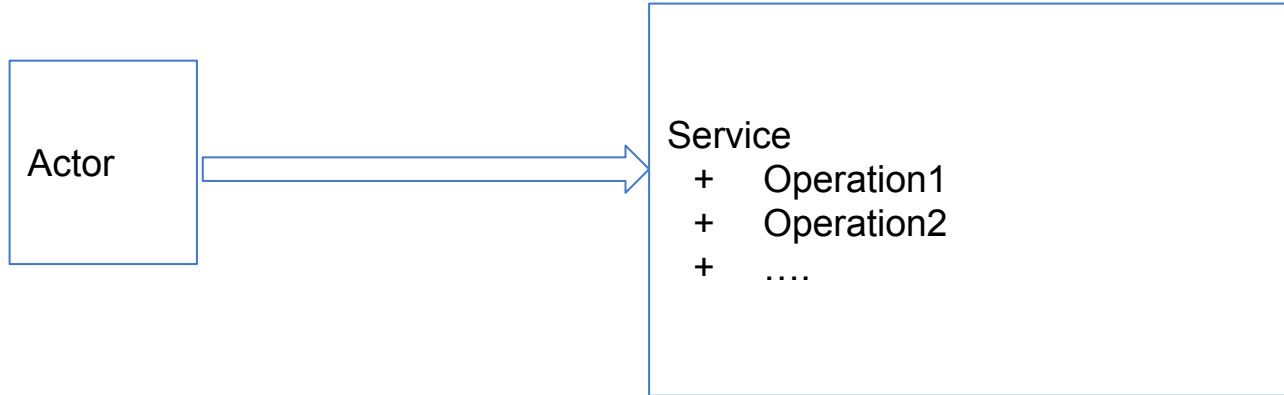
JAVACREAM

*Training
Consulting
Projectmanagement*

Spring Grundlagen

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

Ausgangssituation



Bei Modellierung einer Fachanwendung
keinerlei Bezug zu Spring vorhanden

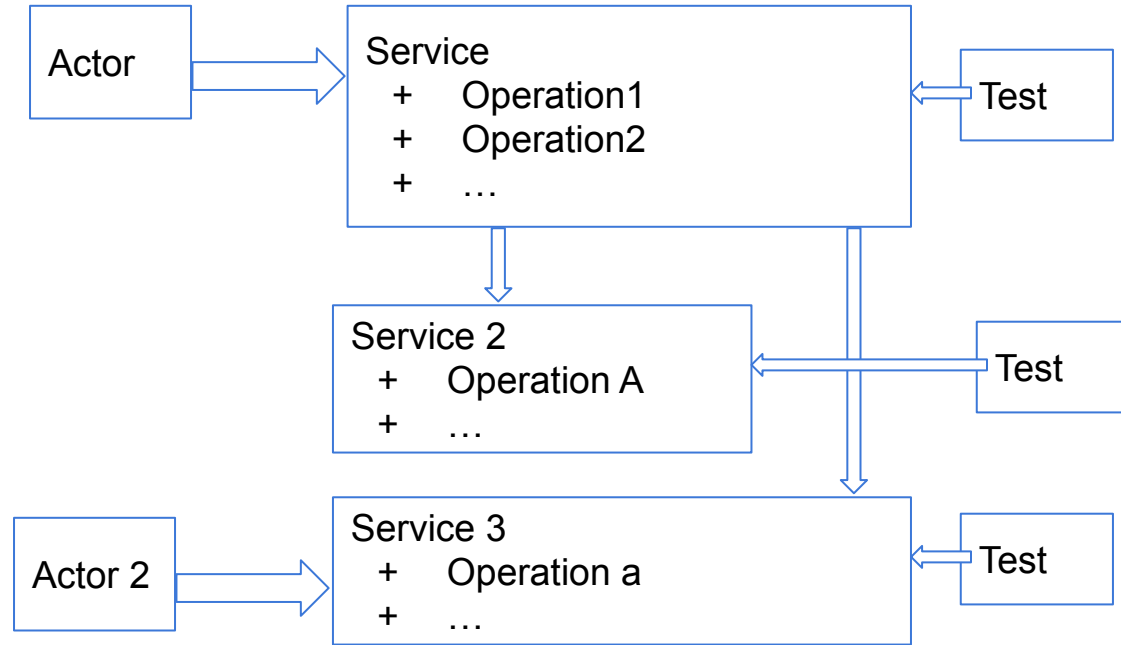
Anforderungen an das Modell

- + Wartbarkeit
- + Wiederverwendung
- + Testbarkeit

Umsetzung durch Modularisierung
statt einer monolithischen
Applikation

Bezug zu Spring ist indirekt

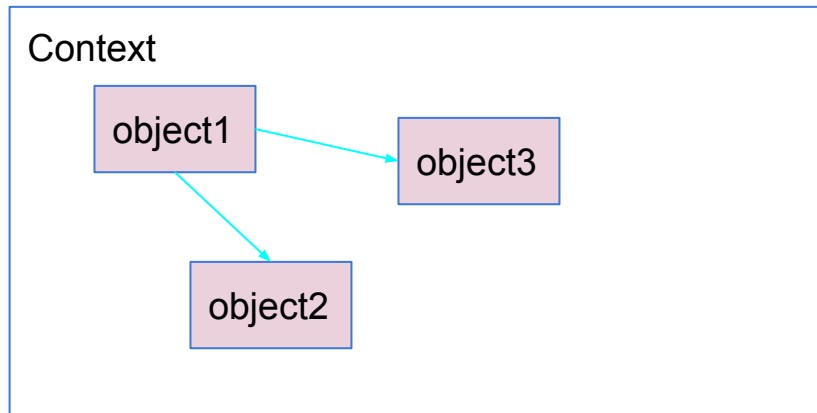
- + Bei Verwendung von Spring ist die Modularisierung eines technischen Modells sehr gut möglich



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”



Programcode der Anwendung bestehend aus Fachklassen

Aufgabe des Contexts

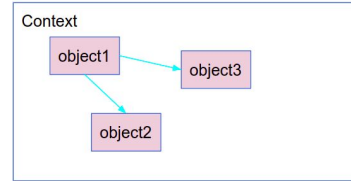
- + Identifikation der relevanten Fachklassen und Instanziierung von Fach-Objekten
- + Identifikation der Abhängigkeiten der Objekte und das Setzen der Abhängigkeit

Service-oriented bzw.
Microservices

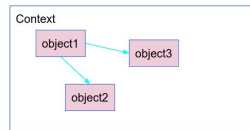
Service 1

zur Laufzeit jeweils ein laufender Prozess

darin läuft ein Spring Context mit Fach-Objekten

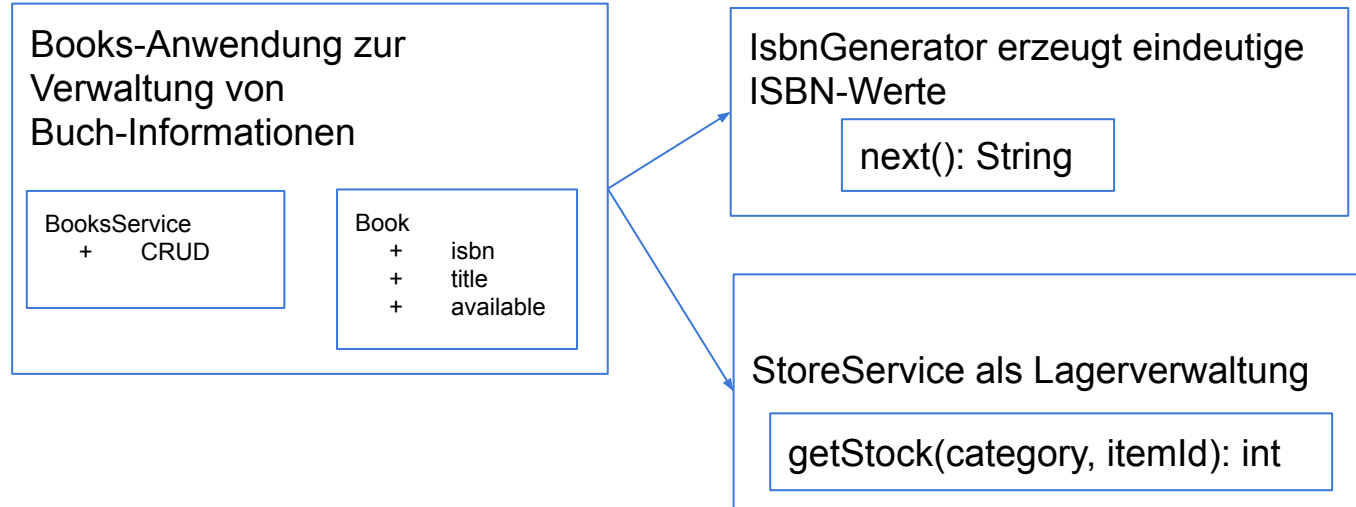


Service 2



Die Fachanwendung des Trainings

- Vollständig vorgegeben
- Fachlich einfach



- 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

- 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 instanziiert
 - 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-Operatoren
 - 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

- 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

- **@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**

- 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”

- 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
-

- 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

- 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-Injection
 - Sind setter-Methoden notwendig?
 - Parameter können auch zur Value-Injection benutzt werden
 - ToDo: RandomIsbnGenerator mit Constructor mit prefix und countryCode