Enterprise Java mit Spring

Eine Praktisch Einführung
EB Zürich

Über Mich

Beruflich

- Software Engineering
- Cloud Enablement Themen
  - Cloud Transformation, DevOps
- Professionalität und Professionalisierung in der IT

Nicht Beruflich

- Snowboarden, Sport

github.com/Vad1mo twitter.com/vad1mo

Kursablauf

- **Jeden Donnerstag - 18:00 - 21:30 - BiZE - Raum 211**
- 13. Juni 2019 18:00 - 21:30
- 20. Juni 2019 18:00 - 21:30
- 27. Juni 2019 18:00 - 21:30
- 4. Juli 2019 18:00 - 21:30
  - Ggf. etwas früher zwecks gemeinsamen Abschlussabend
    - In unmittelbaren Umgebung EB Zürich
    - Pizza, Burger, Sushi, Indisch

Erste Kursabend

- Kennenlernen
- Organisatorisches
- Einführung
- DevEnv Setup
- Hello World!
- Aufgabenstellung
- Depencency Injection und Inversion of Control

Zweiter Kursabend

- Umsetzung der Anforderungen für unsere Beispielanwendung
- Komponenten und die Bedeutung
- Spring MVC
- Testing

Dritter Kursabend

- Neue Anforderungen für unsere Beispielanwendung
  - Persistenz
  - REST API
  - Cloud Deployment

Vierter Kursabend

- Spring Ökosystem
  - Erweiterungen
  - Ergänzungen
- Alternativen zu Spring
- Diskussion
- Abschluss

Referenz und Leitfaden

Spring Boot 2: Moderne Softwareentwicklung mit Spring 5
von Michael Simons

Kursgrundlage

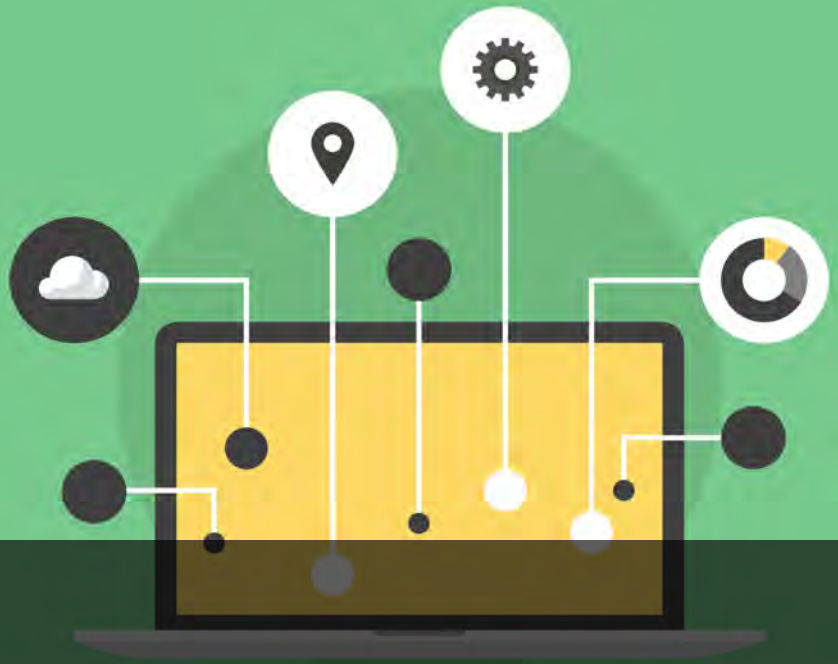
Mit der Buchreferenz erstellen wir eine Anwendung, welche die besprochenen Themen aufgreift und praktisch umsetzt.

Ablauf

- Wechsel zwischen Theorie und Praxis
  - Kein sinnloses „Copy & Paste“
  - Erkenntnis statt Predigt
  - Raum für unterschiedlichen Geschwindigkeiten
- Gegenseitige Unterstützung
- Zwischenfragen erwünscht

- Was sind eure Erwartungen
- Was hofft Ihr heute zu lernen
- Was kennt Ihr schon über Spring


YOUR TURN

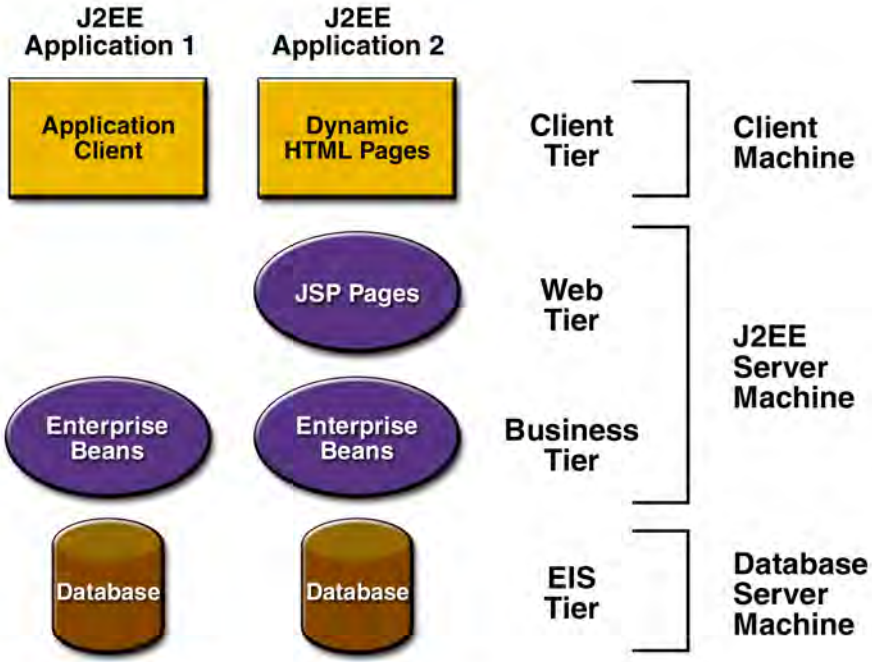


EINFÜHRUNG

Die Geschichte von Spring und Spring Boot

Es war einmal ...

- Java 2 Enterprise Edition (J2EE) in 1999
  - Plattform zur Entwicklung unternehmensweiten verteilten Anwendungen
  - Java Server Pages (JSP), Enterprise Java Bean (EJB), JDBC, JMS, JNDI, Java Transaction API, Java Mail
  - 4 Schichten Architektur
- Sehr Komplex
  - Fragile Konfiguration
  - XML-Hölle
- J2EE Application Server



Die Geschichte von Spring und Spring Boot

Idee

- J2EE Anwendungen ohne EJBs
  - Vereinfachte Entwicklung von Enterprise Java Anwendungen
  - Ergänzend zu J2EE und Java EE
  - Koexistenz mit J2EE und keine Alternative

Erste Veröffentlichung

- 0.9 im Jahr 2003

Kernfunktionen

- Inversion of Control mit Dependency Injection
- AOP
- JDBC, JPA
- MVC-basierte Webanwendungen
- RESTful Services

Was ist Spring?

- Definition ist Kontextbezogen
  - Spring Framework (Core Container)
  - Familie von Komponenten rund um Spring (von Pivotal)
- Das Spring Framework ist in Module unterteilt
  - Core Container
    - Konfigurations Modell und Dependency Injection
  - Messaging
  - Transactionale Daten und Persistenz
  - Servletbasierte Spring MVC Web Framework
  - Reactive web Framework Spring WebFlux
  - Uvm.

Die Geschichte von Spring und Spring Boot

Spring - 11 Jahre Später

- Flexibilität
- Breite Palette von unterstützten Anwendungsszenarien
- Komplexität
  - XML Konfiguration
  - Sehr hohe Einstiegshürde
  - Unübersichtliche Abhängigkeit
  - Wenig Cloudfreundlich

Spring Boot

Spring Boot ist eine neue Sichtweise auf Spring

- Flachere Lernkurve
  - Konvention über Konfiguration gegenüber explizierter Konfiguration
  - Starter Pakete
- Vordefinierte Pakete
- Grösserer Fokus auf Zwölf-Faktoren-App
  - 12factor.net/de/
- Cloud Fokus
- Loslösung von JEE
  - Embedded Container

Spring Aufbau

- Das Spring-Programmiermodell umfasst nicht die Java EE-Plattform-Spezifikation vollständig, sondern integriert ausgewählte Einzelspezifikationen aus dem EE-Dach
  - Servlet API (JSR 340)
  - WebSocket API (JSR 356)
  - Gleichzeitigkeits-Dienstprogramme (JSR 236)
  - JSON Binding API (JSR 367)
  - Bohnenvalidierung (JSR 303)
  - JPA (JSR 338)
  - JMS (JSR 914)
  - sowie ggf. JTA/JCA-Setups zur Transaktionskoordination.

Referenzen

- Cheat Sheet mit wichtigsten Annotation
  - bit.do/spring-boot-cheat
- Spring Referenzdokumentation
  - docs.spring.io/spring/docs/current/spring-framework-reference/
- Spring Boot
  - docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

DEVELOPMENT ENVIRONMENT SETUP

Basic Tooling

- Package Manager für die verschiedenen Tools
- Windows 10
  - Chocolatey, Jabba,
  - Mit WSL - SDKMAN!, Homebrew
- MacOS & Linux
  - SDKMAN!, Homebrew, Jabba
- jabba - cross-platform - Java Version Manager
  github.com/shyiko/jabba

Setup

- Java 11 Installation
  - Windows 10 (Chocolatey, Jabba, SDKman)
  - MacOS/Linux (SDKMAN!, Jabba, Homebrew)

Setup 2

- IDE Installation
  - IntelliJ IDEA
  - Eclipse & Spring Tools 4 for Eclipse
  - Visual Studio Code & Spring Tools 4 for Eclipse
  - Netbeans
- Homebrew
  - `eclipse intellij-idea-ce, intellij-idea`
- Choco
  - `eclipse, intellijidea-community, intellijidea-ultimate`

Checkliste

- ☐ Build Tool – **Maven**, **Gradle**)
  - ☐ Choco, SDKMAN!, Homebrew
- ☐ Docker Installieren
  - ☐ Choco, Homebrew
- ☐ **Spring Boot CLI**
  - ☐ Choco, SDKMAN!, Homebrew
- ☐ **Git**
  - ☐ Choco, Homebrew

Checkliste

```
$ java -version
$ idea
$ mvn -v
$ gradle -v
$ docker --version
$ spring --version
```

- ✓ Java 11
- ✓ IDE
- ✓ Build Tools
  - ✓ Gradle
  - ✓ Maven
- ✓ Docker
- ✓ Spring Boot CLI


HELLO WORLD

Ziel

- Erster Erfolg mit Spring Boot
- Kurzer Blick hinter die Kulissen
- Einfache Anwendung mit einem HTTP Endpunkt
  - Ausgabe vom String `Hello, ${name}` im Browser
  - Wobei «`name`» ein Parameter ist

Maven Projekt Setup

- Verzeichnis anlegen hello-world
- Maven Projekt erstellen
  - `pom.xml` anlegen
  - Inhalt in `pom.xml` kopieren von bit.do/eUToi
- Standardverzeichnisse erstellen für Projektstruktur
  - `src/main/java`
  - `src/main/test`

Maven Project Object Model (POM)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.5.RELEASE</version>
    </parent>
    <groupId>com.example</groupId>
    <artifactId>hello-world</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>hello-world</name>
    <properties...>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build...>
</project>
```

POM Vererbung

GAV von unserem Projekt

Abhängigkeiten

spring-boot-starter-web

spring-boot-starter-test

bit.do/eUToi

Testgetriebener Ansatz

```java
@RunWith(SpringRunner.class)
@WebMvcTest
public class ApplicationTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void expectHelloWorldResponse() throws Exception {
        this.mockMvc.perform(
                get( urlTemplate: "/hello")
                        .param( name: "name", ...values: "World"))
                .andExpect(status().isOk())
                .andExpect(content()
                        .string( expectedContent: "Hello, World\n"));
    }
}
```

Spezieller Runner JUnit

MVC Relevante Konfiguration Mock Umgebung

Injizieren einer Instanz

https://github.com/springbootbuch/helloworld

Unsere Businesslogik

Unsere Anwendung

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Spring Boot Starter Kompositions Annotation

Klasse zum starten der Anwendung

https://github.com/springbootbuch/helloworld

Geheimnis hinter @SpringBootApplication

- @SpringBootApplication ist eine Zusammengesetzte Annotation
- Meta und Composed-Annotationen in Spring
- @SpringBootApplication ist eine Zusammensetzung aus:
  - @SpringBootConfiguration
    - Konfiguration einer Spring Boot Anwendung
  - @EnableAutoConfiguration
    - Ermittelt anhand von Anhängigkeiten welche Komponenten Verfügbar sind und konfiguriert diese
  - @ComponentScan
    - Bestandteil von Spring Core, Sucht im Classpath nach Spring Beans

Was passiert in SpringApplication.run

- Auswahl und Erstellung eines Passenden ApplicationContext
- Registrierung einer CommandLineProperySource
  - Veröffentlichung der Argumete über die CLI an Spring
- Aktualisierung des Kontextes mit allen gefundenen @ Komponenten
- Ausführung aller Beans vom Typ CommandLineRunner

Aufgabe

- Findet einen Weg wie man den ASCII-Art-Banner (beim Starten den Anwendung) ändert könnt.
- Zusatz
  - Logausgabe auf der Konsole nach dem Start der Anwendung

Fazit

- Erster Erfolg mit Spring Boot

Build Management Tools

- Maven oder Gradle
  - Spring Boot Erweiterungen für Maven und Gradle
- Spring Boot liefert ein Set von vordefinierten Abhängigkeiten
  - GAV: org.springframework.boot:spring-boot-starter-parent:2.1.5.RELEASE

Aufgabe
Spring Boot Erweiterung

- Integriert die Spring Boot Maven Erweiterung in euer
  - Build → Plugin aus Pom.xml aus github.com/springbootbuch/helloworld

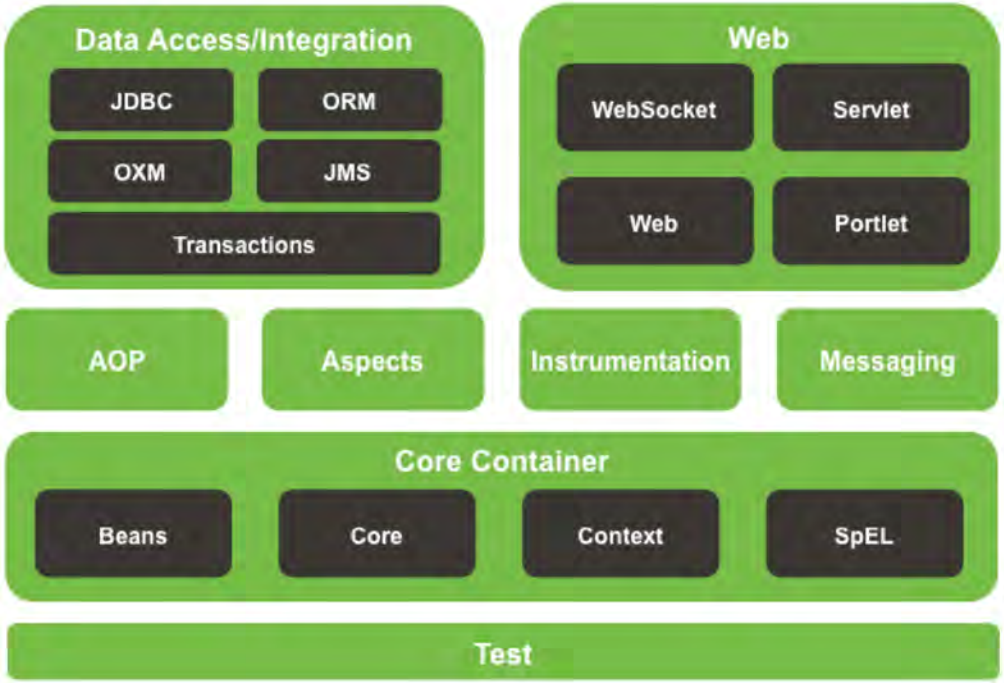
SPRING FRAMEWORK GRUNDLAGEN

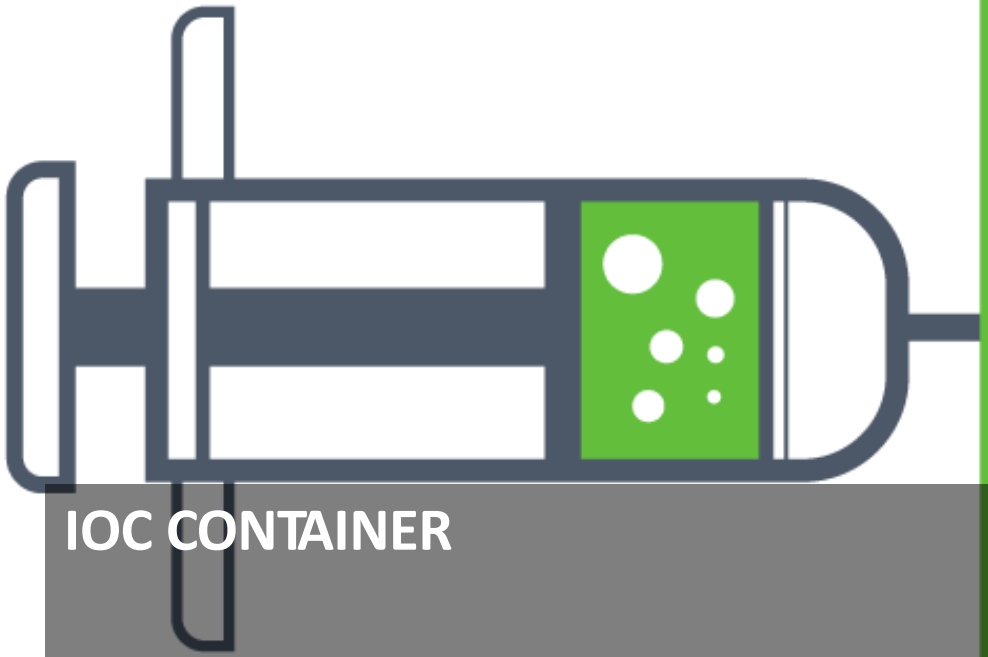
Design-Philosophie

- Auswahl auf allen Ebenen.
- Verzögern der Designentscheidungen auf den spätesten Zeitpunkt.
  - Z.b. Ändern der Persistenzanbieter durch Konfiguration
- Verschiedene Perspektiven
  - Vielzahl von Anwendungsanforderungen mit unterschiedlichen Perspektiven.
- Abwärtskompatibilität

Kernmerkmale

- IoC Container
- Events
- Resources
- I18n
- Validation
- Data Binding
- Type Conversion
- SpEL
- AOP





IOC CONTAINER

Schlüsselelemente

- Wir wollen Softwaresysteme bauen die einfach wartbar und erweiterbar sind
  - Modularisierung der Anwedung erfolgt in der Regel auf folgenden Ebenen
    - Fachlicher Ebene
    - Technischer Eben
- Lose Kopplung
  - Trennung von inhaltlich nicht zusammengehörenden Modulen
  - Einem Modul ist idealerweise unabhängig von einem anderem Modul
- Hohe Kohäsion
  - jede Programmeinheit (Methode, Klasse, Modul) ist verantwortlich für genau eine wohldefinierte Aufgabe bzw. Einheit.
  - Erzielbar mit Software Design,
  - Objektorientierte Programmierung
- Querschnittsaufgaben
  - Nicht funktionaler Natur
  - Security

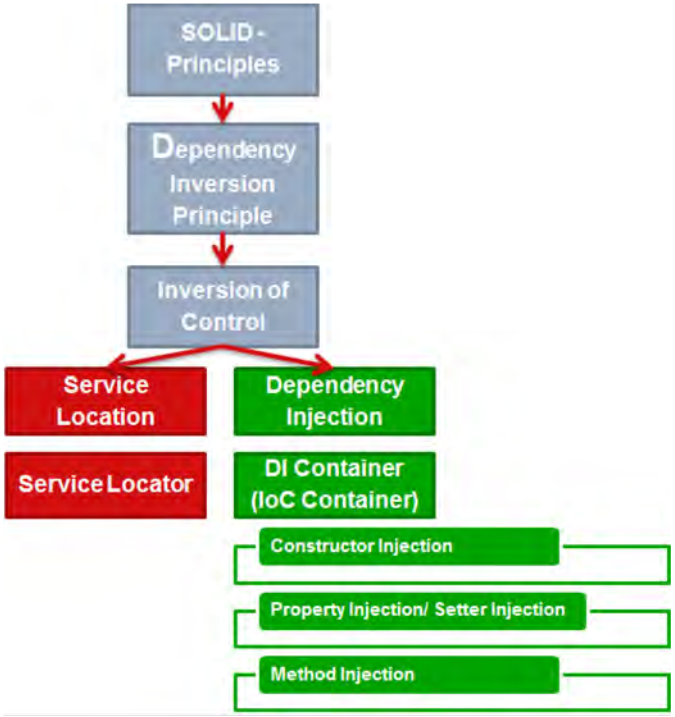
SOLID Prinzip

- **S**ingle Responsibility Prinzip
  - Jede Klasse sollte nur eine einzige Verantwortung haben.
  - Verantwortung wird hierbei als „Grund zur Änderung“ definiert
- **O**pen-Closed Prinzip
  - Module sollten sowohl offen (für Erweiterungen), als auch geschlossen (für Modifikationen) sein.
  - Z.b. Vererbung
- **L**iskovsches Substitutionsprinzip
- **I**nterface Segregation Prinzip
  - Clients sollten nicht dazu gezwungen werden, von Interfaces abzuhängen, die sie nicht verwenden
- **D**ependency Inversion Prinzip

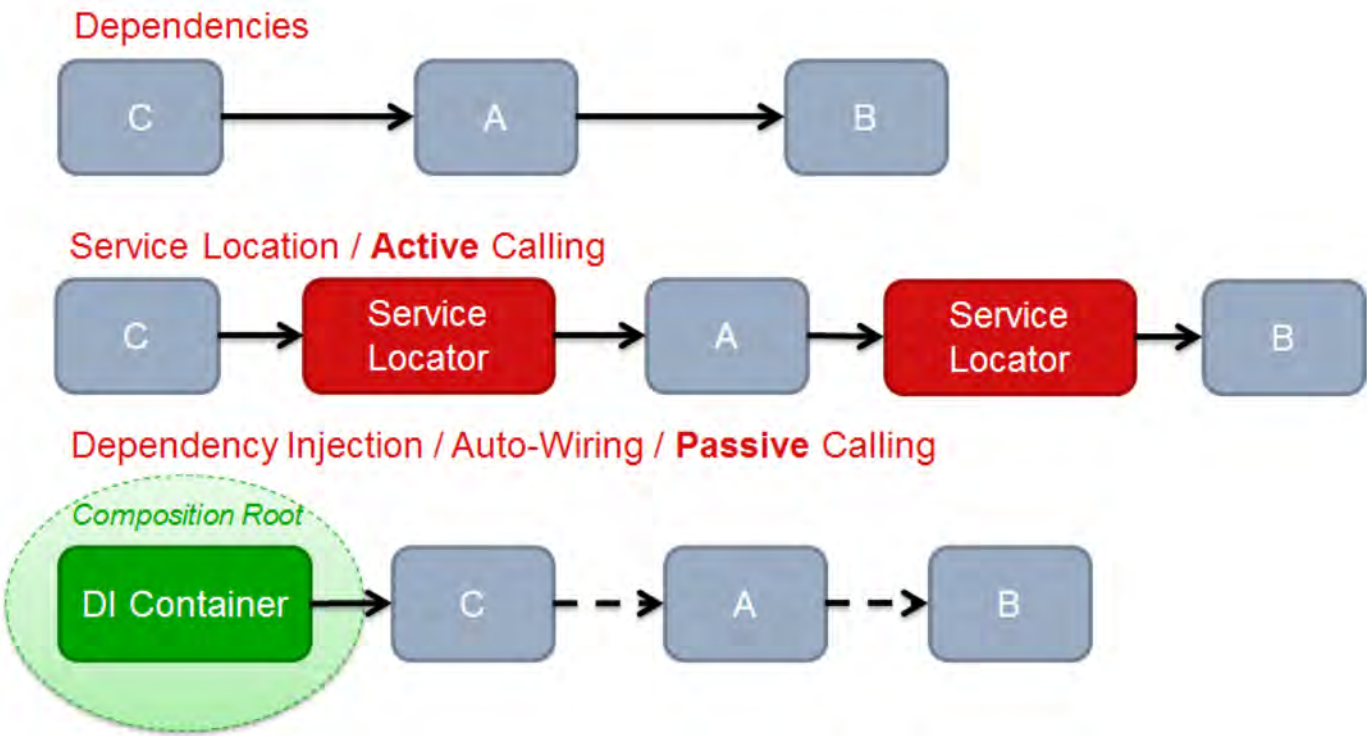
- Einstieg in das Thema
  - de.wikipedia.org/wiki/Prinzipien_objektorientierten_Designs
  - Robert C. Martin Clean Code

Dependency Injection und IOC

- Dependency Injection
  - Teil von IoC Konzept
- Hohe Kohäsion und lose Kopplung von Objekten



Verschiedene Arten Abhängigkeiten und IoC



Spring Dependency Injection

- Lose Kopplung
  - Module die Inhaltlich nicht zusammen gehören sollen möglich lose zueinander stehen
- Hohe Kohäsion
  - Inhaltlich zusammengehörenden Teile sollen

DI als Vorteil in Tests

- Aufgabe:
  - Erstellte drei Klassen A,B,C die voneinander abhängen.
    - A-->B-->C
    - Jeder Klasse hat eine Methode a,b,c
    - Jede Methode hat einen zusammengesetzten Rückgabewert aus a, b und c
    - Beispiel a.a() --> a b c
  - Schreibt einen Test für A und versucht B und C zu isolieren.
  - Versuch das Gaze in Spring zu implementieren wobei A,B,C @Components sind

```
@Test
public void testA() {
    String result = new A().a();
    assertEquals(result, actual: "a b c");
}
```

AOP – Aspekt Orientierte Programmierung

- eine Möglichkeit, dem bestehenden Code Verhalten hinzuzufügen, ohne diesen Code zu ändern.
- Querschnittsaufgaben von oft nicht funktionaler Natur
  - Caching
  - Security
  - Transaktionen

```
@Transactional
public void orderGoods(Order order) {
    // A series of database calls to be performed in a transaction
}
```

Spring Container

- Module des Spring Containers
  - Normale Jar-Libraries
  - spring-core, spring-beans
    - Grundlegende Funktionalitäten DI und IOC
  - spring-context, spring-context-support
    - ApplicationContext und Anwendungsspezifische Funktionen
    - Einstiegspunkt in das Spring Framework
    - @SpringApplication
  - spring-expression
- Teil von spring-boot-starter

Bean

- Beans sind die von Spring-Container verwalteten Objekte
  - Vom Spring Container instanziiert und konfiguriert
  - Vollständige Verwaltung von Spring-Container
- Spring Container Verwaltet die Abhängigkeiten

Spring Scopes

- Unterschiedliche Scopes für Beans
- Singleton (default)
  - Genau eine Bean
  - Eine einzelne Bean-Definition auf eine einzelne Objektinstanz pro Spring IoC-Container.
- Prototype
  - Zustandsbehaftet
  - Bei Zugriff neu Instanziert
  - Umfasst eine einzelne Bean-Definition mit einer beliebige Anzahl von Objektinstanzen
- Request, Session, globalSession, application
  - Zusätzliche Scopes im Web Context

Spring Konfiguration

- Annotationen
  - Bevorzugte Methode in Spring Boot und in vielen Beispielen
- Java basiert
  - Bevorzugte Methode
  - Konfiguration von Spring in Java @Configuration @Bean
- XML
  - Konfiguration von Spring mit einer XML
  - Legacy
- Kombination aller Möglichkeiten


AUFGABE

bookit.page Vision

- bookit.page ermöglicht allen Menschen einfach und komfortabel etwas zu online zu reservieren.
- Unsere Kunden sind alle Unternehmen bei denen es üblich ist einen Termin zu vereinbaren.
  - Restaurant, Friseur, Arzt oder Hundesalon
- Unser USP (Alleinstellungsmerkmal)
  - Im Gegensatz zu unserem Wettbewerbern ist unser Dienst kostenlos.
  - Wie verdienen Geld durch Added-Value Services

bookit.page Reservation Plattform

- Als Startup möchten wir eine neuartige Online Restaurant Reservation Platform auf dem Markt bringen
- Wenig Budget
- Knappes Zeitfenster
- Early Release to Market
- MVP Ansatz
- Jeder Release bring neue Funktionen
- Unsere Go-to-Market Strategie
  - Zielgruppe sind Restaurants die Ihren Gästen die Möglichkeit bieten möchten einen Tisch Online zu reservieren.

Aufgabe

- Erzeugt eine Webanwendung dies es ermöglicht online einen Tisch zu reservieren.
  - MVP-Ansatz (Minimum Viable Product)
    - Was müssten wir alles weglassen damit es immer noch möglich ist für den Kunden einen Tisch zu reservieren und unser Produkt zu verkaufen.
- Eine Webanwendung
  - Erstellt eine neue Webanwendung
  - HTML Formular um in einem Restaurant ein Tisch zu reservieren
  - Übermittlung der Formularinformationen
    - E-Mail
  - Speichern

Vorgehen

- Git init
- Neue Spring Anwendung Erstellen
  - Web Starter
  - Mail Starter
- Implementierung
  - Controller
  - Web View
- Unit Tests

Neues Projekt Erstellen

- Spring Initializr
  - CLI Zum Erstellen von Spring Projekten
  - Alternative zur DIE
- Spring Boot Funktion in IDEs
  - Project -→ New
- Website start.spring.io
  - Baukasten für neue Anwendungen

```
$ spring
usage: spring [--help] [--version]
 <command> [<args>]
Available commands are: run [options] <files> [--] [args] Run a spring groovy script ...
more command help is shown here
```