Lernskript: Spring Core - IoC & Dependency Injection

Das Herz des Spring Frameworks ist nicht eine Bibliothek, sondern ein Design-Prinzip: **Inversion of Control (IoC)**, das durch den Mechanismus **Dependency Injection (DI)** umgesetzt wird.

Teil 1: Die grundlegende Analogie - Das "Warum"

- **Konzept:** Anstatt dass ein Objekt seine Abhängigkeiten (andere Objekte, die es zum Arbeiten braucht) selbst erstellt, deklariert es nur, was es benötigt. Eine externe Instanz – der **Spring IoC Container** – ist dafür verantwortlich, diese Abhängigkeiten zu erstellen und sie dem Objekt zur Verfügung zu stellen ("zu injizieren").
- **Analogie (Der Restaurant-Manager):**
  - **Traditionell:** Der `Koch` holt sich sein `Messer` selbst aus der Schublade. Er ist fest an ein spezifisches Messer gebunden.
  - **Mit Spring (IoC):** Die **Kontrolle** wird umgekehrt. Der `Restaurant-Manager` (Spring Container) ist für die Werkzeuge verantwortlich, nicht der Koch.
  - **Mit Spring (DI):** Der Koch sagt nur: "Ich brauche ein `Messer`." Der Manager nimmt ein passendes Messer aus seinem Lager und **injiziert** es in den Gürtel des Kochs. Der Koch ist dadurch flexibel und muss nicht wissen, wie das Messer hergestellt wird.

Teil 2: Praktische Anwendungsfälle - Das "Wofür"

Use Case 1: Constructor Injection (Der Goldstandard)

- **Zweck:** Der sicherste und klarste Weg, zwingend erforderliche Abhängigkeiten bereitzustellen.
- **Vorteil:** Das Objekt kann nicht ohne seine Abhängigkeiten erstellt werden und ist somit immer in einem gültigen Zustand. Die Abhängigkeiten können `final` deklariert werden.
- **Code-Beispiel:**

```
@Service // Markiert diese Klasse als Spring Bean
public class NotificationService {
    private final MessageSender sender; // Die Abhängigkeit

    // Spring benutzt diesen Konstruktor für die Dependency Injection
    @Autowired
    public NotificationService(MessageSender sender) {
        this.sender = sender;
    }
}
```

Weitere Injection-Typen (seltener verwendet)

```
Setter Injection: Über eine set...-Methode. Nützlich für optionale Abhängigkeiten.

Field Injection: Direkt am Feld (@Autowired private ...). Sollte vermieden werden, da es Abhängigkeiten versteckt, das Testen erschwert und keine final-Felder erlaubt.
```

Teil 3: Wichtige Regeln & Vertiefung (Profi-Tipps)

```
Stereotyp-Annotationen: Um eine Klasse für Spring sichtbar zu machen, markiert man sie mit einer Annotation.

    @Component: Generische Spring-Komponente.

    @Service: Für die Business-Logik-Schicht.

    @Repository: Für die Datenbank-Zugriffs-Schicht.

    @Controller / @RestController: Für die Web-Schicht (lernen wir später).

Was ist ein "Spring Bean"? Ein "Bean" ist der Begriff für ein Objekt, das vom Spring IoC Container instanziiert und verwaltet wird. Jede Klasse mit einer Stereotyp-Annotation wird zu einem Bean.

@Qualifier (Wenn es mehrere Kandidaten gibt):

    Problem: Es gibt zwei Beans, die dasselbe Interface implementieren (z.B. EmailSender und SmsSender). Spring weiß nicht, welches es injizieren soll.

    Lösung: Man gibt den Beans Namen (@Component("email")) und fordert mit @Qualifier("email") explizit das richtige an.
    code Java
```

public NotificationService(@Qualifier("email") MessageSender sender) { this.sender = sender; }