

LU02b - Refactoring und Optimierungen 1

Prinzipien

- **DRY (Don't Repeat Yourself)**
- **SRP (Single Responsibility Principle)**
- *OCP (Open/Closed Principle)*
- *LSP (Liskov Substitution Principle)*
- *ISP (Interface Segregation Principle)*
- *DIP (Dependency Inversion Principle)*
- *KISS (Keep It Simple, Stupid)*
- *YAGNI (You Ain't Gonna Need It)*
- *SoC (Separation of Concerns)*
- *Law of Demeter (Principle of Least Knowledge)*

SOLID steht wiederum für SRP, OCP, LSP, ISP, DIP

Wir fokussieren uns erstmal auf **DRY** und **SRP**.

DRY

Code ist viel simpler art- und erweiterbar, wenn man nur eine einzige Stelle anpassen muss. Duplizierte Code-Blöcke erhöhen die Fehleranfälligkeit und erschweren die Lesbarkeit.

Wenn Sie zum Beispiel an zwei unterschiedlichen Orten denselben Datenbankaufruf haben, kann man diese Funktionalität auslagern in eine gemeinsame Methode (oder oftmals gar eine eigene Klasse).

```
try (Connection conn = DriverManager.getConnection(URL, USER,
PASSWORD);
    ...
} catch (SQLException e) {
    ...
}
```

```
public class Database {
    private static final String URL = Config.get("DB_URL");
    private static final String USER = Config.get("DB_USER");
    private static final String PASSWORD = Config.get("DB_PASSWORD");

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

Kontrollieren Sie Ihren Code hinsichtlich dieses Prinzips und korrigieren Sie bei Bedarf.

SRP

Jede Klasse sollte nur einen Aufgabenbereich haben. Das Lesen und Schreiben von Dateien ist somit zu trennen von der Logik für den Zugriff auf die Datenbank usw.

Kontrollieren Sie Ihren Code hinsichtlich dieses Prinzips und korrigieren Sie bei Bedarf.

Design-Patterns

Es gibt zahlreiche Design-Patterns. Diese werden im späteren Verlauf des Semesters anschauen.

Creational Design Patterns	Structural Design Patterns	Behavioral Design Patterns
Factory Method	Adapter	Chain of Responsibility
Abstract Factory	Bridge	Command
Builder	Composite	Iterator
Prototype	Decorator	Mediator
Singleton	Facade	Memento
	Flyweight	Observer
	Proxy	State
		Strategy
		Template Method
		Visitor

Mehr dazu auf <https://refactoring.guru/design-patterns>

From:
<https://wiki.bzz.ch/> - **BZZ** - Modulwiki

Permanent link:
<https://wiki.bzz.ch/de/modul/ffit/3-jahr/java/learningunits/lu02/scanner?rev=1756157561>

Last update: **2025/08/25 23:32**

