SOLID –Verstehen und Anwenden

Simon Wagner, Kristin Utech, Sabine Neubauer (andrena objects ag)

Ablauf & Organisatorisches

1. Setup
2. Vorstellung und Diskussion der SOLID-Prinzipien
3. Programmieren
    - bis zu 4 Übungsaufgaben
    - nach jeder Übung Vorstellung und Diskussion der Lösungen
    - Pausen bei Bedarf

Setup

1. Finde einen Pairing-Partner, der die Übungen in der gleichen Sprache durchführen möchte
2. Richtet eure Entwicklungsumgebung ein
3. Source-Code
   - auf USB-Stick oder
   - unter https://github.com/emilybache/Racing-Car-Katas/tree/master/Java

SOLID

1. **S**ingle Responsibility Principle
2. **O**
3. **L**
4. **I**
5. **D**

Single Responsibility Principle

„Es sollte nie mehr als einen Grund dafür geben, eine Klasse zu ändern.“

- Robert C. Martin

Single Responsibility Principle

Beispielcode

SOLID

1. **S**ingle Responsibility Principle
2. **O**pen/Closed Principle
3. **L**
4. **I**
5. **D**

Open/Closed Principle

„Module sollten sowohl offen (für Erweiterungen) als auch geschlossen (für Modifikationen) sein.“

- Bertrand Meyer

Open/Closed Principle

Beispielcode

SOLID

1. **S**ingle Responsibility Principle
2. **O**pen/Closed Principle
3. **L**iskov Substitution Principle
4. **I**
5. **D**

Liskov Substitution Principle

„Sei q(x) eine Eigenschaft des Objektes x vom Typ T, dann sollte q(y) für alle Objekte y des Typs S gelten, wobei S ein Subtyp von T ist.“

- Barbara Liskov

Liskov Substitution Principle

„Es besagt, dass ein Programm, das Objekte einer Basisklasse T verwendet, auch mit Objekten der davon abgeleiteten Klasse S korrekt funktionieren muss, ohne dabei das Programm zu verändern.“

- Wikipedia

Liskov Substitution Principle

„Subtypes must be substitutable for their base types.“

- Robert C. Martin

Liskov Substitution Principle

Beispielcode

SOLID

1. **S**ingle Responsibility Principle
2. **O**pen/Closed Principle
3. **L**iskov Substitution Principle
4. **I**nterface Segregation Principle
5. **D**

Interface Segregation Principle

„Clients sollten nicht dazu gezwungen werden, von Interfaces abzuhängen, die sie nicht verwenden.“

- Robert C. Martin

Interface Segregation Principle

Beispielcode

SOLID

1. **S**ingle Responsibility Principle
2. **O**pen/Closed Principle
3. **L**iskov Substitution Principle
4. **I**nterface Segregation Principle
5. **D**ependency Inversion Principle

Dependency Inversion Principle

„A. Module hoher Ebenen sollten nicht von Modulen niedriger Ebenen abhängen. Beide sollten von Abstraktionen abhängen.

B. Abstraktionen sollten nicht von Details abhängen. Details sollten von Abstraktionen abhängen.“

- Robert C. Martin

Dependency Inversion Principle

Beispielcode

SOLID

1. **S**ingle Responsibility Principle
2. **O**pen/Closed Principle
3. **L**iskov Substitution Principle
4. **I**nterface Segregation Principle
5. **D**ependency Inversion Principle

Übung 1 - TirePressureMonitoringSystem

1. **Aufgabe**:
   - Schreibe Unit-Tests für die Klasse `Alarm`
   - Überlege, welche SOLID-Prinzipien verletzt sind
   - Refaktorisiere, so dass die SOLID-Prinzipien (besser) erfüllt werden
2. **Kontext:**
   - Der Reifendruck wird überwacht
   - Falls der Druck außerhalb des erwarteten Bereichs liegt soll es einen Alarm geben
   - `Sensor`-Klasse liefert zufällige aber realistische Werte
   - Verhalten der `Sensor`-Klasse soll **nicht** geändert werden

Übung 2 - TextConverter

1. **Aufgabe**:
   - Schreibe Unit-Tests für die Klasse `HtmlTextConverter`
   - Überlege, welche SOLID-Prinzipien verletzt sind
   - Refaktorisiere, so dass die SOLID-Prinzipien (besser) erfüllt werden
2. **Kontext:**
   - Eine Textdatei soll als HTML formatiert um den Inhalt in einem Browser anzeigen zu können
3. **Sonstiges:**
   - Zusatzaufgabe: `HtmlPagesConverter`
   - Der `HtmlPagesConverter` unterstützt zusätzlich zur Formatierung noch Paging

Übung 3 - TurnTicketDispenser

1. **Aufgabe**:
    - Schreibe Unit-Tests für die Klasse `TicketDispenser`
    - Überlege, welche SOLID-Prinzipien verletzt sind
    - Refaktorisiere, so dass die SOLID-Prinzipien (besser) erfüllt werden
2. **Kontext:**
    - In einer Behörde sollen Nummern gezogen werden
    - Die Nummern werden nach und nach aufgerufen
    - Es gibt mehrere Ticket Dispenser in der Behörde
    - Zwei Personen sollen nie die gleiche Nummer ziehen können

Übung 4 – Telemetry System

1. **Aufgabe**:
   - Schreibe Unit-Tests für die Klasse `TelemetryDiagnosticControls`
   - Überlege, welche SOLID-Prinzipien verletzt sind
   - Refaktorisiere, so dass die SOLID-Prinzipien (besser) erfüllt werden
2. **Kontext:**
   - Der Telemetry Server soll angefragt werden um von diesem aktuelle Diagnose-Informationen zu erhalten
   - `TelemetryClient` ist von außen vorgegeben und kann nicht geändert werden

Quellen

1. Racing Car Kata auf github bei Emily Bache: https://github.com/emilybache/Racing-Car-Katas
2. Racing Car Kata auf github bei Luca Minudel: https://github.com/lucaminudel/TDDwithMockObjectsAndDesignPrinciples/tree/master/TDDMicroExercises

Quellen

1. http://t3n.de/news/prinzipien-software-entwicklung-solid-615556/
2. http://code.tutsplus.com/series/the-solid-principles--cms-634
3. Software Craftmansship Berlin: https://drive.google.com/folderview?id=0B9kuBIdCsSLYfk1MTGZud1BoQ2czY3ppSl9wTTlGMXJkWEU3S1F1Y3FaVlNJOGNobm5TdU0&usp=sharing
4. Wikipedia: https://de.wikipedia.org/wiki/Dependency-Inversion-Prinzip
5. Wikipedia: https://de.wikipedia.org/wiki/Liskovsches_Substitutionsprinzip