

10. Tutorium

Best Practices, JUnit, Einsicht

Tutorium 14

Péter Bohner | 25.01.2022



Inhaltsverzeichnis

1. Best Practices

2. JUnit

3. Präsenzübung

4. Aufgabe

5. Ende

Best Practices
○○○○

JUnit
○○○○○○

Präsenzübung
○

Aufgabe
○

Ende
○

Polymorphie vs. **instanceof**

Falls ihr Code der folgenden Art schreibt:

```
if (obj instanceof Type1) {  
    // do something code1  
} else if (obj instanceof Type2) {  
    // do something code2  
}
```

kann dieser *meistens* durch vernünftige Polymorphie ersetzt werden:

- Oberklasse von Type1 und Type2 hat (*abstrakte*) Methode doSomething()
- Unterklasse Type1 überschreibt doSomething() mit *code1*
- Unterklasse Type2 überschreibt doSomething() mit *code2*
- ...
- Fallunterscheidung fällt weg:
 - obj.doSomething()
 - richtige Methode wird durch dynamische Bindung ausgeführt

`==` vs. `equals()`

`==` vs. `equals()`

`==` vergleicht:

- Inhalt von primitiven Datentypen (**boolean**, **char**, **int**, **double**, ...)
- Referenzen auf Objekte/Arrays

`equals()` vergleicht:

- Inhaltliche Gleichheit zweier Objekte

Achtung

Standardimplementierung von `equals()` in `Object` vergleicht auch nur Referenzen!

⇒ überschreibe `equals()`, falls nötig.

`equals()` Implementierung Vorgehen

Wichtige Fragen bei der Implementierung:

- **Benötigt die Klasse überhaupt eine `equals()`-Methode?**
 - Nur wenn die Objekte der Klassen logisch miteinander verglichen werden können und müssen, d.h. wenn der Vergleich zweier Referenzen nicht genügt.
- **Welche Objekte sollen verglichen werden?**
 - **Nur Objekte derselben Klasse?**
 - nutze `getClass()`
 - **Objekte der Unterklassen mit Objekten der Basisklasse?**
 - nutze `instanceof`
 - Nur möglich, falls die abgeleitete Klasse nur das Verhalten der Basisklasse ändert
 - keine zusätzlichen Attribute kommen zum Vergleich hinzu (Symmetrie-Eigenschaft erhalten)
 - Deklariere `equals()`-Methode als `final`, um sicherzustellen, dass keine Attribute zum Vergleich in Unterklassen hinzukommen

equals() für Objekte der selben Klasse

Daumenregel

Sobald eine Basisklasse die equals()-Methode implementiert, muss die equals()-Methode der abgeleiteten Implementierung **super.equals()** aufrufen.

(Ausnahme: Basisklasse ist java.lang.Object)

```
public class Person {
    private String name;
    private String lastName;

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj != null && getClass() == obj.getClass()) {
            Person person = (Person) obj;
            return this.name.equals(person.name)
                && this.lastName.equals(person.lastName);
        }
        return false;
    }
}
```

```
public class GermanCitizen extends Person {
    private int nationalIdNumber;

    @Override
    public boolean equals(Object obj) {
        if (super.equals(obj)) {
            GermanCitizen citizen = (GermanCitizen) obj;
            return this.nationalIdNumber
                == citizen.nationalIdNumber;
        }
        return false;
    }
}
```

Automatisiertes Testen

- Programme müssen getestet werden
- Vergleich von gewünschtem und tatsächlichem Verhalten
- Hilft bei der Fehlersuche nach Änderungen
- Hier: Komponententests
 - Methoden
 - Input/Output
 - Algorithmen
- Testen mit JUnit
 - Testen einzelner Methoden
 - Verschiedene Fälle werden getestet (Äquivalenzklassen, Grenzwerte)

Best Practices
○○○○

JUnit
●○○○○○

Präsenzübung
○

Aufgabe
○

Ende
○

Installation

- **Eclipse** Standardmäßig installiert
- **IntelliJ** Standardmäßig installiert

Anwendung

- **Eclipse**
 - Rechtsklick auf gewünschten Paket/Projekt → New → JUnit Test Case (oder Other → Java → JUnit → JUnit Test Case) → Klassennamen wählen
- **IntelliJ**
 - Gewünschte Klasse des Projektes öffnen → Code → Generate → Test → Klassennamen wählen

Mehr zu JUnit unter Eclipse im Wiki:

https://ilias.studium.kit.edu/goto.php?target=wiki_1275181_Testen_mit_JUnit

Beispiel

```
7 public class ClassTest {  
8  
9     private Class testobject;  
10    private static final int FIRST_SUMMAND = 10;  
11    private static final int SECOND_SUMMAND = 5;  
12  
13    @Before  
14    public void setUp() {  
15        this.testobject = new Class();  
16    }  
17
```

```
18    @Test  
19    public void testSum() {  
20        int result = testobject.getSum(FIRST_SUMMAND,  
21            SECOND_SUMMAND);  
22        assertEquals(FIRST_SUMMAND + SECOND_SUMMAND,  
23            result);  
24    }  
25  
26    @After  
27    public void tearDown() {  
28        this.testobject = null;  
29    }  
30  
31 }
```

JUnit - Annotationen

■ @BeforeClass, @AfterClass

- statische Methode läuft einmal zu Beginn bzw. am Ende der Testklasse

■ @BeforeEach, @AfterEach

- Methode läuft jeweils zu Beginn bzw. am Ende eines einzelnen Tests

■ @Test

- Methode ist eine Test-Methode

■ @Test (expected = `ArrayIndexOutOfBoundsException.class`)

- Es wird erwartet, dass die Test-Methode eine bestimmte Exception wirft

■ @Ignore

- Der Test wird beim Testen ignoriert

JUnit - Assert Methoden

Nutze Assertions, um Ergebnisse von Methoden zu überprüfen

- `assertEquals(Object expected, Object current)`
- `assertEquals(double expected, double actual, double delta)`
- `assertFalse(boolean result), assertTrue(boolean result)`
- `fail(), fail(String message)`
- `assertNotNull(Object current), assertNotSame(Object expected, Object current)`

Wichtig

- Mehrere Assertions in einem Test erlaubt
- `org.junit.Assert` enthält Assertions
- einzelne Tests klein halten \Rightarrow erleichtert Fehlersuche

Beispielinteraktionen testen

- eigene Test-Klasse schreiben, die Input-Datei mit Output-Datei vergleicht
- Mit diff/FC arbeiten
 - In der Kommandozeile in den Ordner navigieren, der den ersten Paketordner enthält
 - `javac paketstruktur/*.java`
 - Linux:
`java paketstruktur/Main < sample.in | diff - sample.out`
 - Windows Cmd:
`java paketstruktur/Main < sample.in > output.txt`
`FC sample.out output.txt`
 - ⇒ Fehler suchen

Statistiken

- Teilnahmen: 21/24
- avg: 7,86, med: 9.0, stdev: 2.64
- Bestanden mit: ≥ 6.0 Punkten

Ablauf

- Studierendenausweis und Lichtbildausweis bereithalten
- Kein Schreibzeug oder elektronische Geräte auf dem Tisch
- Das anfertigen von Kopien, Abschriften, Fotos, oä ist untersagt

Aufgabe

Verschlüsselung

Modelliert ein abstraktes Chiffre (*eng. cipher*). Was muss ein Cipher können?
Implementiert konkret ein Caesar-Chiffre, und falls ihr Zeit habt ein Ersetzungschiffre.
Schreibt JUnit Tests für eure Klasse(n).

Caesarchiffre

Rotiert Buchstaben *key* Stellen im Alphabet weiter, alle anderen Zeichen werden nicht verändert.

Ersetzungschiffre

Ersetzt in der Eingabe bestimmte Character mit anderen. z.B. Ersetze *a* mit *z* und *2* mit *!*. Dann wird aus
abc123 *zbc1!3*

Bis zum nächsten Tutorium am 01.02.2022.

Best Practices
○○○○

JUnit
○○○○○○

Präsenzübung
○

Aufgabe
○

Ende
●