OOSE 3

Testen mit BlueJ und JUnit

Lehrstuhl Softwaretechnologie, Dipl.-Inf. Martin Morgenstern
Sommersemester 2019

"Test Your Software, or Your Users Will."

The Pragmatic Programmer

"Testing shows the presence of bugs, but never their absence."

Edsger W. Dijkstra

Vorgehen beim Unit-Test allgemein

1. Testfälle ausdenken / Testfalltabellen erstellen
2. Testfälle nach gemeinsamen Fixtures (Halterungen) in Klassen gruppieren
3. i. Allg. pro Testklasse eine Halterung (Fixture)
   - `@Before`-annotierte Methode / **setUp()**
   - `@After`-annotierte Methode / **tearDown()**
   - pro Testfall eine Testmethode schreiben
4. Testfälle (nach Änderungen im Programm wiederholt) ausführen → Regressionstest

Wie wähle ich Testdaten für Testfälle aus (Vorlesung!)

- Bestimme die Extremwerte der Parameter der zu testenden Methode (**Grenzwertanalyse**)
  - Nullwerte immer testen, z.B. 0 oder `null`
  - Randwerte z.B. 1.1. und 31.12.
- Bestimme Bereichseinschränkungen
  - Werte außerhalb eines Zahlenbereiches
  - Negative Werte, wenn natürliche Zahlen im Spiel sind
- Bestimme **Äquivalenzklassen** von Testdaten und teste nur die Repräsentanten
- Bestimme **Zustände**, in denen sich ein Objekt nach einer Anweisung befinden muss
- Bestimme alle Werte aller **booleschen Bedingungen** in der Methode (Steuerfluss)

Äquivalenzklassenbildung (1)

Anforderungen genau definieren

Beispiel:

- Basispreis für Flug von A nach B kostet 100 €.
- Buchung im Reservierungssystem für Reihe 1-10 kostet 20 € mehr , Reihe 11-21 kosten 10 € weniger. Das Flugzeug hat nur 21 Sitzreihen.
- Essen kostet 5 € extra.
- Wenn Frühbucher (bis 4 Wochen vor Flugtermin), dann 10 % Rabatt auf Gesamtpreis, sonst kein Rabatt auf Buchung.

Äquivalenzklassenbildung (2)

Bildung der Äquivalenzklasse durch Klassifizierung der Wertebereiche für Ein- und Ausgabedaten

Beispiel

- **Parameter 1: Basispreis**
  ÄK11={100€}
- **Parameter 2: Reihe // Das Flugzeug hat nur 21 Reihen.**
  ÄK21={1-10}
  ÄK22={11-21}
- **Parameter 3: Verpflegung // Ja - Nein**
  ÄK31={keine Verpflegung}
  ÄK32={Verpflegung}
- **Parameter 4: Rabatt // 2 Rabattierungstypen**
  ÄK41={bis 4 Wochen vor Termin}
  ÄK42={kein Rabatt}

Äquivalenzklassenbildung (3)

Definition von Testfällen für

1. **gültige Äquivalenzklassen**

   Beispiel

   – 100% Äquivalenzklassenabdeckung (Vollständige Abdeckung)

   → 1*2*2*2 = 8 Testfälle

2. **ungültige Äquivalenzklassen**

   Beispiel

   – Ungültige Sitzreihen

   – Ungültiger Rabattierungstyp

Testfalltabelle für Buchung: Fehlerfälle

Nr.	Eingabe			Ausgabe calculatePrice()	Erwarteter Status
	int row	boolean wantsMeal	int weeksAgo		
B1	**0**	false	0		Exception (IAE)
B2	**-1**	false	0		Exception (IAE)
B3	**22**	false	0		Exception (IAE)
B4	1	false	**-1**		Exception (IAE)

IAE ... IllegalArgumentException

Testfalltabelle für Buchung: Gutfälle

Nr.	Eingabe			Ausgabe calculatePrice()	Erwarteter Status
	int row	boolean wantsMeal	int weeksAgo		
B5	10	false	3	120.0	OK
B6	11	false	3	90.0	OK
B7	10	true	3	125.0	OK
B8	11	true	3	95.0	OK
B9	10	false	4	108.0	OK
B10	11	false	4	81.0	OK
B11	10	true	4	112.5	OK
B12	11	true	4	85.5	OK

Verwendung von *Annotationen* um

- Methoden als Testmethoden zu kennzeichnen
- die Testabarbeitung zu konfigurieren

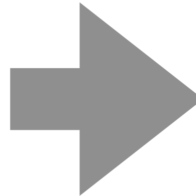
Empfehlenswertes Tutorials

- Junit 4 allgemein: http://www.frankwestphal.de/JUnit4.0.html
- BlueJ / JUnit4: http://www.bluej.org/tutorial/testing-tutorial.pdf

Annotation	Description
@Test public void method()	The @Test annotation identifies a method as a test method.
@Test (expected = Exception.class)	Fails if the method does not throw the named exception.
@Test(timeout=100)	Fails if the method takes longer than 100 milliseconds.
@Before public void method()	This method is executed before each test. It is used to prepare the test environment (e.g., read input data, initialize the class).
@After public void method()	This method is executed after each test. It is used to cleanup the test environment (e.g., delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures.
@BeforeClass public static void method()	This method is executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as static to work with JUnit.
@AfterClass public static void method()	This method is executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as static to work with JUnit.
@Ignore	Ignores the test method. This is useful when the underlying code has been changed and the test case has not yet been adapted. Or if the execution time of this test is too long to be included.

Herkunft von assertEquals() und Co. in JUnit 4

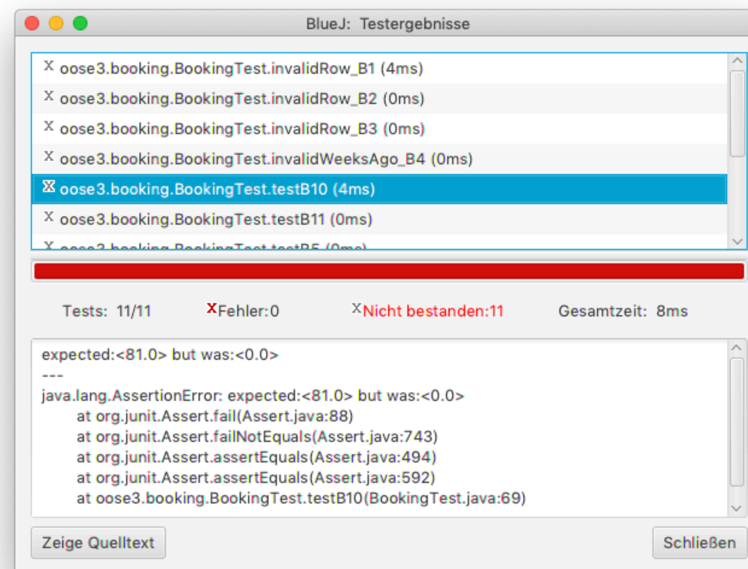
```java
package oose3;

import org.junit.Assert;
import org.junit.Test;

public class ExampleTest {
     @Test
     public void someTest() {
          Assert.assertEquals(12, 3 * 4);
     }
}
```



```java
package oose3;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class ExampleTest {
     @Test
     public void someTest() {
          assertEquals(12, 3 * 4);
     }
}
```

DEMO mit BlueJ

- Test Driven Development: wir schreiben erst die JUnit-Tests
- Danach implementieren wir den Code, sodass die Tests grün werden



"Flüssige" Assertions mit AssertJ

- JUnit 4 Assert: assertEquals(wert1, wert2)
  - An welcher Stelle kommt der erwartete Wert (expected)?
  - Keine Helfer für "enthält String xyz" oder "hat Größe x"
  - Schlechte Lesbarkeit
- AssertJ: assertThat(wert).isEqualTo(erwarteterWert)
  - Liest sich wie ein Satz
  - Keine Vertauschung von expected/actual möglich
  - Enthält Helfer für Listen, Strings, usw.
    - assertThat("abcdef").contains("cd")
  - Assertions verkettbar (daher auch "fluent API" genannt)
    - assertThat("abcdef").hasSize(6).endswith("def")
- Kombinierbar sowohl mit JUnit 4 oder 5

TECHNISCHE UNIVERSITÄT DRESDEN

org.junit
Class Assert

```
java.lang.Object
  └org.junit.Assert
```

```
public class Assert
extends java.lang.Object
```

A set of assertion methods useful for writing tests. Only failed assertions are recorded. These methods can be used directly: `Assert.assertEquals(...)`, however, they read better if they are referenced through static import:

```
import static org.junit.Assert.*;
   ...
   assertEquals(...);
```

See Also:
`AssertionError`

Constructor Summary

protected	`Assert()` Protect constructor since it is a static only class

Method Summary

static void	`assertArrayEquals(byte[] expecteds, byte[] actuals)` Asserts that two byte arrays are equal.
static void	`assertArrayEquals(char[] expecteds, char[] actuals)` Asserts that two char arrays are equal.
static void	`assertArrayEquals(int[] expecteds, int[] actuals)` Asserts that two int arrays are equal.
static void	`assertArrayEquals(long[] expecteds, long[] actuals)` Asserts that two long arrays are equal.
static void	`assertArrayEquals(java.lang.Object[] expecteds, java.lang.Object[] actuals)` Asserts that two object arrays are equal.
static void	`assertArrayEquals(short[] expecteds, short[] actuals)` Asserts that two short arrays are equal.
static void	`assertArrayEquals(java.lang.String message, byte[] expecteds, byte[] actuals)` Asserts that two byte arrays are equal.
static void	`assertArrayEquals(java.lang.String message, char[] expecteds, char[] actuals)` Asserts that two char arrays are equal.
static void	`assertArrayEquals(java.lang.String message, int[] expecteds, int[] actuals)` Asserts that two int arrays are equal.

org.assertj.core.api

Class Assertions

java.lang.Object
    org.assertj.core.api.Assertions

Direct Known Subclasses:

`BDDAssertions`

```
public class Assertions
extends Object
```

Entry point for assertion methods for different types. Each method in this class is a static factory for a type-specific assertion object.

For example:

```
int removed = employees.removeFired();
assertThat(removed).isZero();

List<Employee> newEmployees = employees.hired(TODAY);
assertThat(newEmployees).hasSize(6);
```

This class only contains all `assertThat` methods, if you have ambiguous method compilation error, use either `AssertionsForClassTypes` or `AssertionsForInterfaceTypes` and if you need both, fully qualify you assertThat method.

Java 8 is picky when choosing the right `assertThat` method if the object under test is generic and bounded, for example if foo is instance of T that extends Exception, java 8 will complain that it can't resolve the proper `assertThat` method (normally `assertThat(Throwable)` as foo might implement an interface like List, if that occurred `assertThat(List)` would also be a possible choice - thus confusing java 8.

This why `Assertions` have been split in `AssertionsForClassTypes` and `AssertionsForInterfaceTypes` (see http://stackoverflow.com/questions/29499847/ambiguous-method-in-java-8-why).

Author:

`Alex Ruiz, Yvonne Wang, David DIDIER, Ted Young, Joel Costigliola, Matthieu Baechler, Mikhail Mazursky, Nicolas François, Julien Meddah, William Delanoue`

Constructor Summary

Constructors

Modifier	Constructor and Description
`protected`	`Assertions()` Creates a new `Assertions`.

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

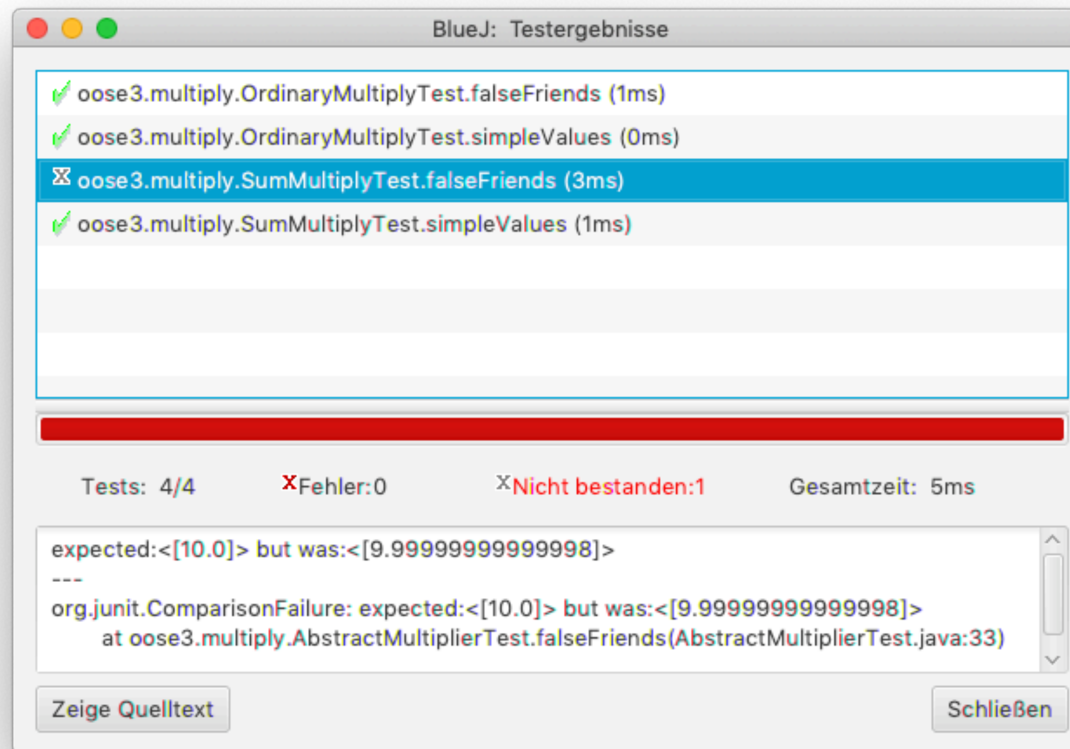
Modifier and Type	Method and Description
`static <T> Condition<T>`	`allOf(Condition<? super T>... conditior` Creates a new `AllOf`

Fallstricke bei Floating-Point-Zahlen

- Vergleich *a* == *b* muss durch |*a* - *b*| <= *e* ersetzt werden, wobei e die zulässige Genauigkeit beschreibt
- JUnit 4 ~~Assert.assertEquals(double expected, double actual)~~ daher deprecated (= "ausgemustert")
    - stattdessen: assertEquals(double expected, double actual, **double delta**)
- AssertJ: assertThat(x).isCloseTo(y, **within(delta)**)
- DEMO: Multiplier-Projekt in BlueJ

TECHNISCHE UNIVERSITÄT DRESDEN

Fallstricke bei Floating Point Zahlen



Vererbung von Testklassen in der Multiplier-Demo

- `AbstractMultiplierTest` enthält die eigentlichen Testfälle
  - Instanziiert aber keine konkrete Implementierung von `Multiplier`
- … dies passiert erst in den `@Before`-Methoden von
  - `SumMultiplyTest`
  - `OrdinaryMultiplyTest`
- Attribut `mult` muss daher Sichtbarkeit `protected` haben