ASMIQ

JUnit 5 Workshop

Mehr Spass und weniger Bauchschmerzen beim Entwickeln durch clevere Tests

Michael Inden

CTO@ASMIQ AG

Speaker Intro

- Michael Inden, Year of Birth 1971
- Diploma Computer Science, C.v.O. Uni Oldenburg

- ~8 ¼ Years @ Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Years @ IVU Traffic Technologies AG in Aachen
- ~4 ¼ Years @ Zühlke Engineering AG in Zürich
- Since June 2017 @ **Direct Mail Informatics / ASMIQ** in Zürich

- Author @ dpunkt.verlag

E-Mail: michael.inden@asmiq.ch
Courses: https://asmiq.ch/
Blog: https://jaxenter.de/author/minden

Agenda

Workshop Contents

ASMIQ

- **PART 1: Einstieg**
  - Warum testen?
  - Einfluss von / auf Qualität
  - Gute Angewohnheiten

- **PART 2: JUnit 5 Intro**
  - Architektur
  - First Test
  - Tests mit mehreren Asserts
  - Testing Exceptions
  - Tests mit Timeouts

Workshop Contents

ASMIQ

- **PART 3: JUnit 5 Advanced**
  - Parameterized Tests
  - Repeated Tests
  - Nested Tests
  - Simple Extensions
- **PART 4:** Tips zur Migration JUnit 4 => JUnit 5
  - AssertJ
- **PART 5:** Test Smells

PART 1:
Warum testen und gute Angewohnheiten

Warum testen wir?

ASMIQ

- Gewünschtes Verhalten beschreiben
- Funktionalität prüfen (auch Randfälle)
- Sicherheitsnetz aufbauen
- Qualitätssicherung
- Kundenzufriedenheit
- weniger Ärger, Nerven und mehr Spass

Warum testen wir? Was ist Qualität?

- Unter **Qualität** versteht jeder leicht etwas anderes:
  - gute Benutzbarkeit oder
  - umfangreiche Funktionalität.
  - Immer jedoch hohes Maß an **Erwatungskonformität** und **Zuverlässigkeit** gewünscht
  - Im richtigen Leben erwarten wir fast immer eine Qualität von **nahezu 95 – 100 %**.

- **Wieso geben wir uns bei Software oftmals mit weniger zufrieden**? Was könnten die Ursachen nicht optimaler Qualität sein?
  - Komplexität ist in Software häufig recht hoch
  - Einzelteile sind zum Teil nicht gut getestet
  - Informatik hat noch nicht den Ingenieursgrad wie Autoindustrie oder Maschinenbau

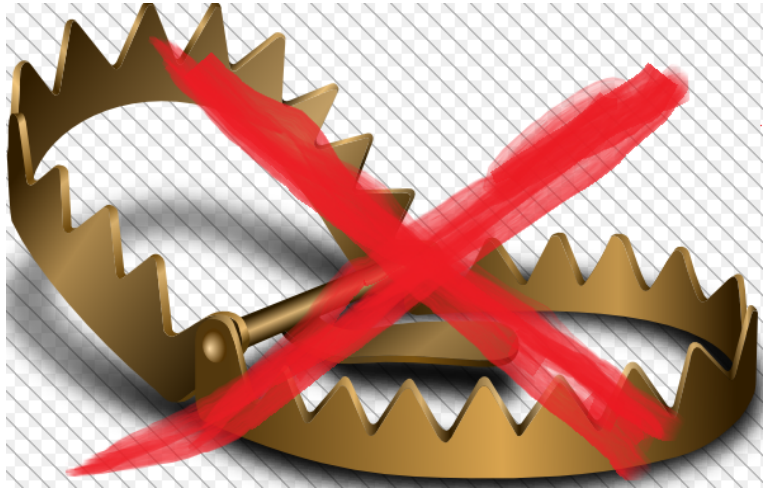
External Quality

External Quality entspricht Benutzersicht

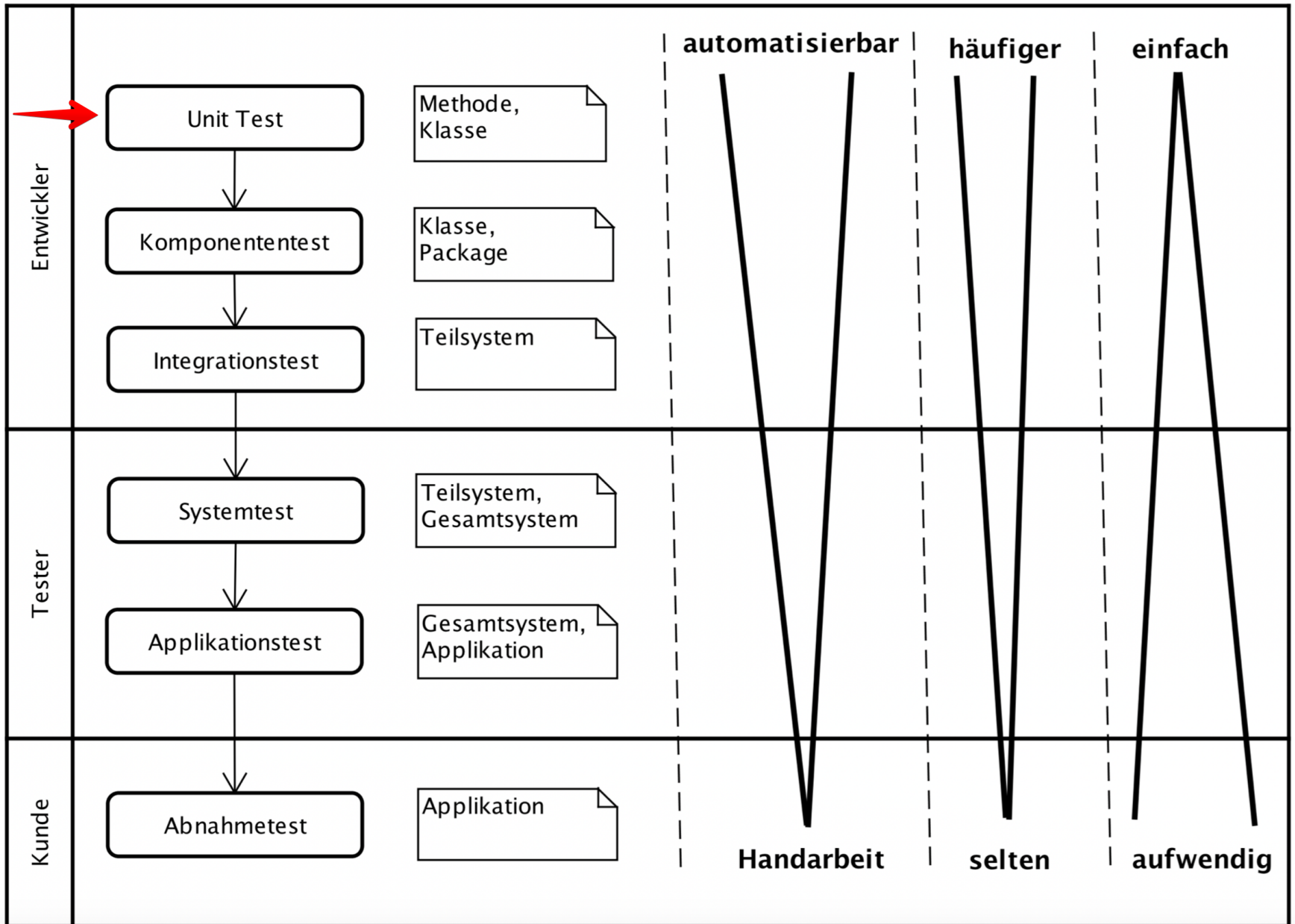
- Arbeitet wie erwartet
- Stellt alle benötigte Funktionalität bereit
- Korrektheit
  - Nahezu keine (beobachtbaren) Bugs
  - Gut getestet
- Benutzbar
- Verlässlich
- …

Internal Quality ~ Entwicklersicht (Code, Build, Testing):

- Lesbarkeit
- Verständlichkeit
- Keine Fallstricke und Hindernisse
- Erweiterbar, pflegbar
- Gut/sinnvoll dokumentiert
- Gute Test/Code Coverage
- …



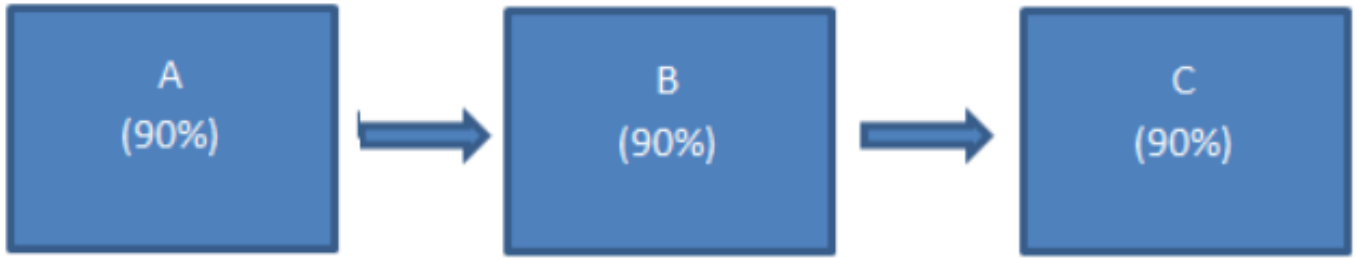
Arten von Tests



Einfluss der Qualität der Einzelteile

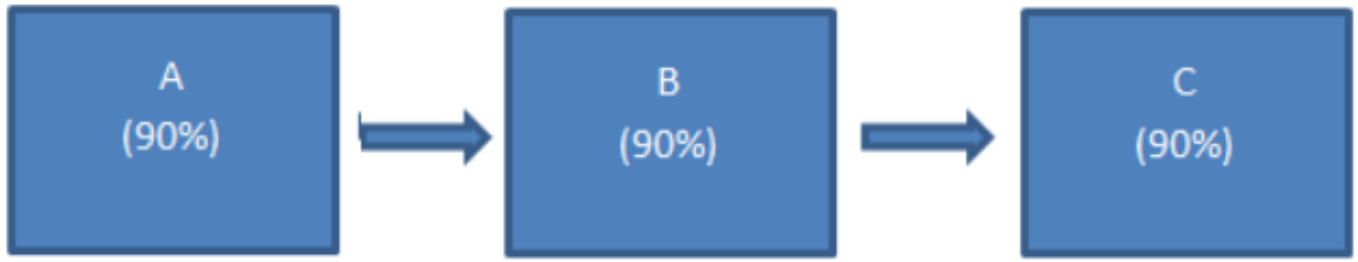
- Die Industrie besitzt **Normen** und **Standards**. Qualität sowie Interoperabilität ist gegeben.

- Je höher die Qualität der Einzelteile, desto höher ist auch die Qualität des Gesamtprodukts. Schauen wir auf ein einfaches System mit drei Bausteinen:



- Wie hoch ist die Gesamtqualität?

Einfluss der Qualität der Einzelteile



- Laut **Systemtheorie**: Produkt der Einzelqualitäten

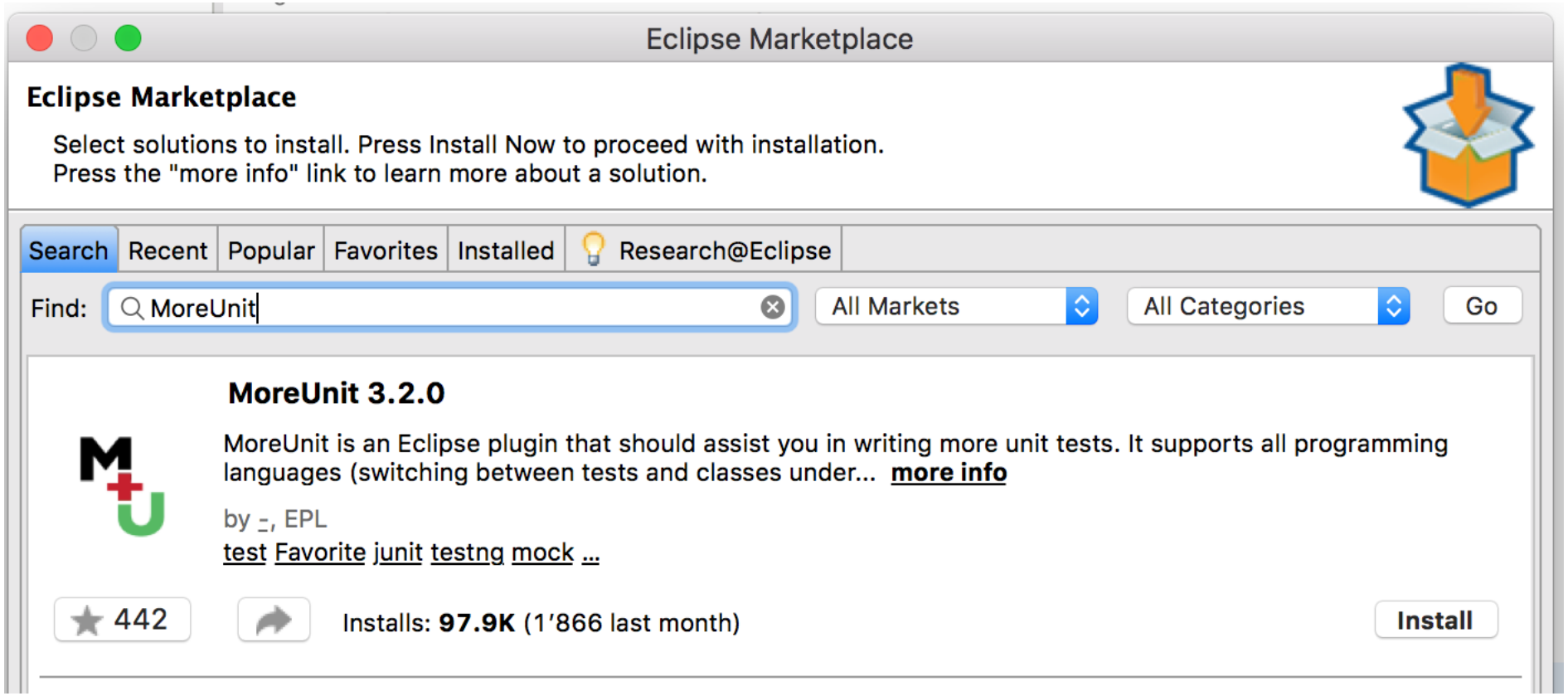
$$0.9 * 0.9 * 0.9 = 0.73 \Rightarrow 73\ \%$$

- Auf Software übertragen, haben wir dabei folgende Probleme:
  - Wer hat schon einmal Systeme mit nur 3 Klassen oder Bestandteilen gebaut?
  - Normale Anwendungen bestehen nicht nur aus einer Vielzahl an Klassen und Objekten, sondern diese besitzen insbesondere auch komplizierte, teils verzwickte Abhängigkeiten.

Gute Angewohnheiten

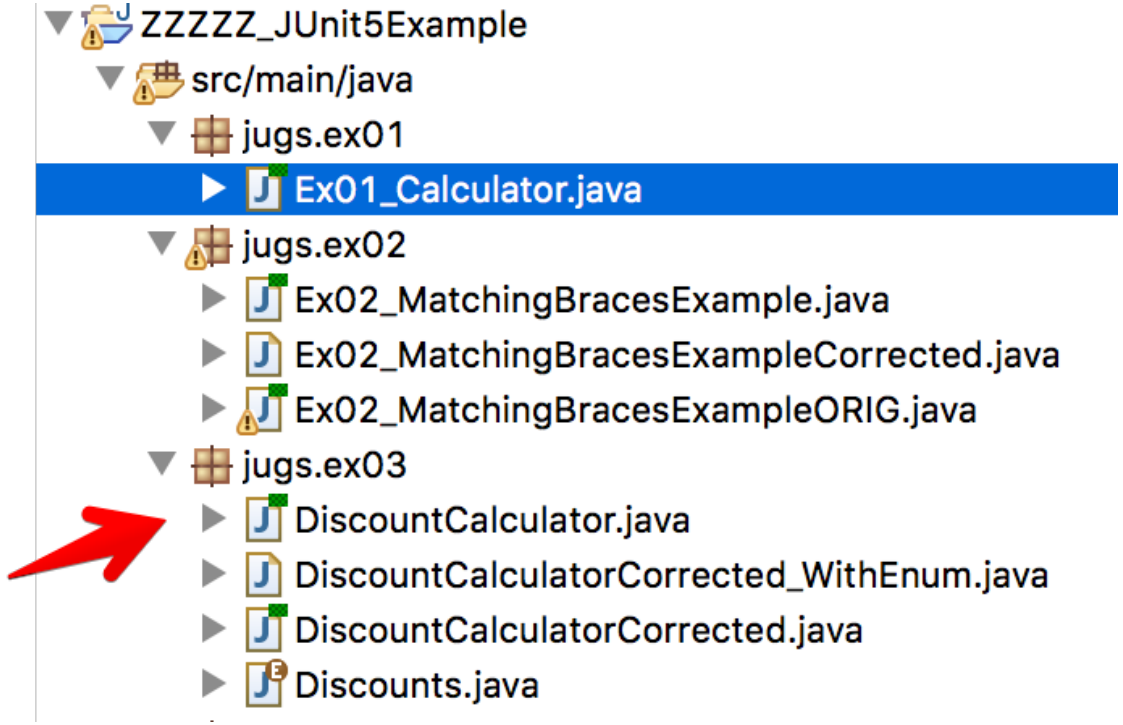
(Eclipse) Plugin MoreUnit

- http://moreunit.sourceforge.net/
- http://moreunit.sourceforge.net/update-site/



(Eclipse) Plugin MoreUnit

- Tastaturkürzel zum Ausführen (CTRL+R) und zum Wechseln zwischen Klasse und Test (CTRL+J).
- Icon-Dekoration im Package Explorer: grünen Punkt zeigt, ob zu einer Klasse ein Test existiert.
- Refactorings: Klassen und korrespondierende Testklassen werden synchron zueinander verschoben oder umbenannt.



Namensgebung

- Klasse `Abc` => zugehörige Testklasse `AbcTest`

- Methoden:
  - Optional: Kürzel `test` als Start
  - Sinnvolle Beschreibung des Testfalls:
  - Methodenname, Bedingungen und Ergebnis im Namen kodieren => **CamelCase wird oft unleserlich**
  - Testing Guru Roy Osherove schlägt Folgendes vor

```
MethodName_StateUnderTest_ExpectedBehavior
MethodName_ExpectedBehavior_WhenTheseConditions
```

```
calcSum_WithValidInputs_ShouldSumUpAllValues()
calcSum_ThrowsException_WhenNullInput()
```

Testfälle definieren

ASMIQ

- Unit Tests **prüfen kleine Bausteine**, meistens Klassen oder Methoden
- **Möglichst isoliert** und ohne Interaktion mit anderen Komponenten
- Tests werden **in Form von Methoden** implementiert
- Im Idealfall: Für jede ***relevante*** **Applikationsmethode** mindestens **eine Testmethode**
- Eine Testmethode **prüft genau eine Funktionalität** (oder nur einen Teil davon) , **idealerweise nur 1 ASSERT !**
- Testmethoden **kurz, klar und verständlich** halten
- **ABER: Wie erreicht man das?**

AAA-Stil

- **ARRANGE** - **ACT** – **ASSERT** (Auch GWT genannt für GIVEN – WHEN – THEN)
- **ARRANGE**: Vorbedingungen und Initialisierungen (***Testfixture***)
- **ACT**: danach wird eine Aktion ausgeführt
- **ASSERT**: Prüfen, ob der erwartete Zustand eingetreten ist

```
@Test
void listAdd_AAAStyle()
{
   // GIVEN: An empty list
   final List<String> names = new ArrayList<>();

   // WHEN: adding 2 elements
   names.add("Tim");
   names.add("Mike");

   // THEN: list should contain 2 elements
   assertEquals(2, names.size(), "list should contain 2 elements");
}
```

FAIR - Gewünschte Eigenschaften von Unit Tests

F – Fast, Focussed

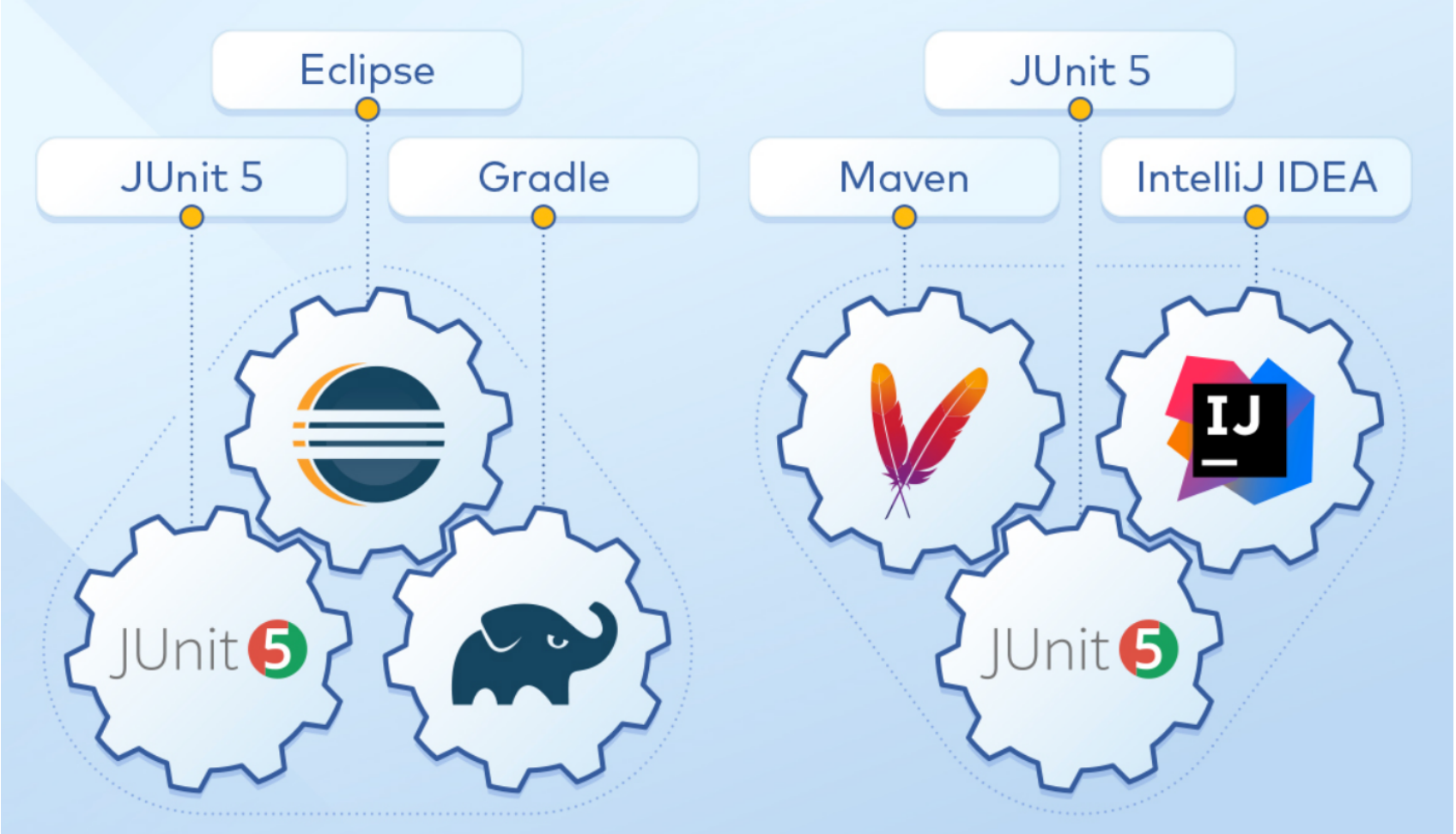
A - Automated

I - Isolated

R – Reliable, Repeatable

Tests (regelmässig) ausführen

- mvn clean test
- gradle clean test
- Eclipse (MoeUnit): Ctrl + R
- IntelliJ: Ctrl + Shift + R



https://www.toptal.com/java/getting-started-with-junit

PART 2:
JUnit 5 Intro

JUnit 5

JUnit 5

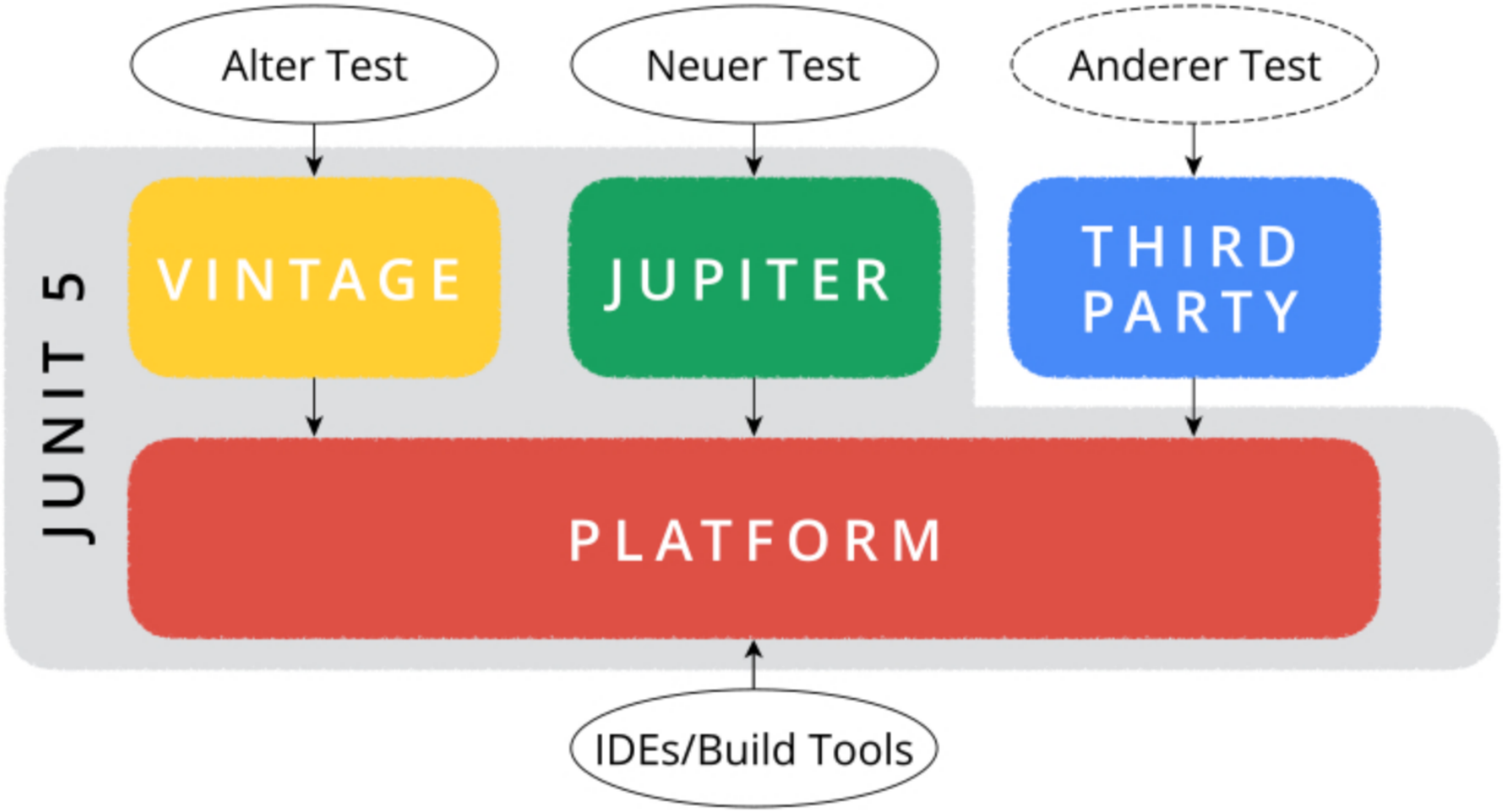
JUnit 4

The new major version of the programmer-friendly testing framework for Java

User Guide | Javadoc | Code & Issues | Q & A | Support JUnit

Architektur

- JUnit 5 =
  JUnit Platform +
  JUnit Jupiter +
  JUnit Vintage



https://jaxenter.de/junit-5-beyond-testing-framework-81787

Assertions -- Bedingungen prüfen

- Auswertung von Bedingungen – Die Klasse **`Assert`** (JUnit4) / **Assertions** (JUnit 5) stellt eine Menge von Prüfmethoden bereit, mit denen Bedingungen formuliert und dadurch **Zusicherungen** über den zu testenden Sourcecode geprüft werden können:

  - **`assertEquals()`** – zwei Objekte auf inhaltliche Gleichheit (Aufruf von **equals(Object)**) bzw. zwei Variablen primitiven Typs auf Gleichheit prüfen*

  - **`assertTrue()`** und **`assertFalse()`** – boolesche Bedingungen prüfen

  - **`assertNull()`** bzw. **`assertNotNull()`** – Objektreferenzen auf == null bzw. != null prüfen
  - **`assertSame()`** bzw. **`assertNotSame()`** – Objektreferenzen auf == bzw. != prüfen

  - **`fail()`** – einen Testfall bewusst fehlschlagen lassen

*) Achtung für Floating Point: float und double

Ein erster Unit Test mit JUnit 5

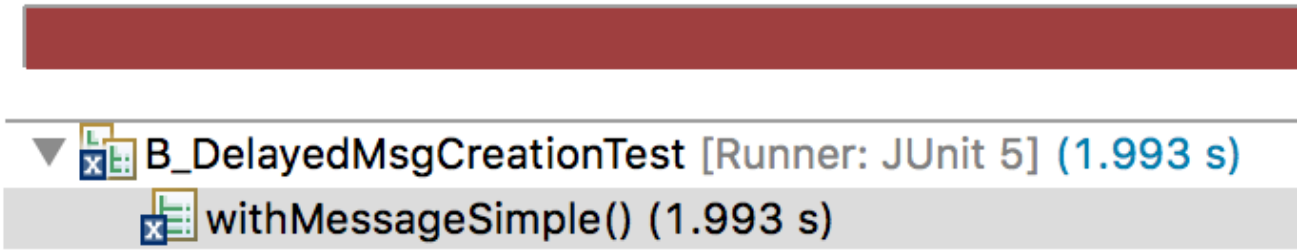
ASMIQ

- Testfälle in Form spezieller Testmethoden erstellt, die mit der Annotation `@Test` markiert

```
@Test
void assertMethodsInAction()
{
    String expected = "Tim";
    String actual = "Tim";
    assertEquals(expected, actual);
    assertEquals(expected, "XYZ", "Hint if wrong");

    assertTrue(true);
    assertTrue(true, "Always true");

    assertFalse(false);
    assertNull(null);
    assertNotNull(new Object());
    assertSame(null, null);
    assertNotSame(null, new Object());
}
```

Komplexe Erzeugung von Messages

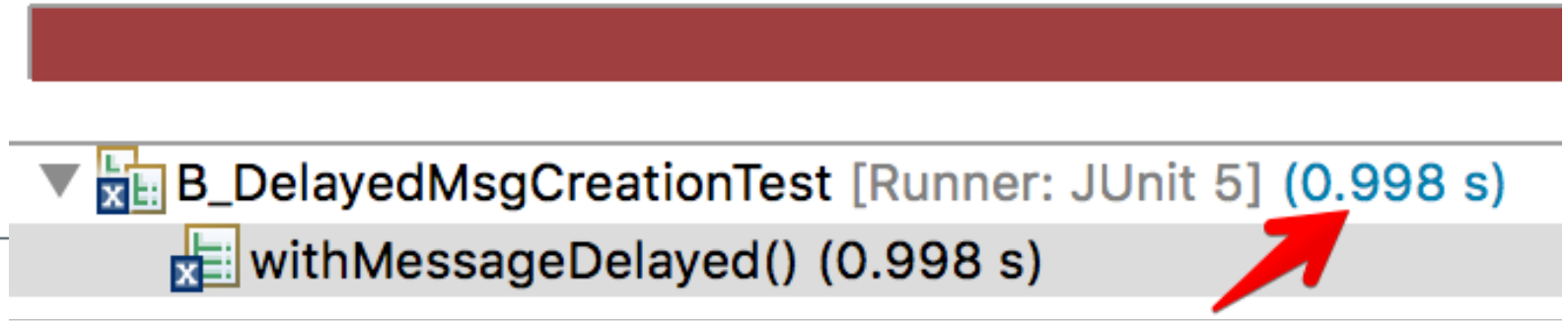
```
@Test
void withMessageSimple()
{
    String expected = "Tim";

    assertEquals(expected, "Tim", complicatedCalculation("Hint"));
    assertEquals(expected, "ALWAYS", complicatedCalculation("Hint"));
}

private String complicatedCalculation(String info)
{
    try
    {
        Thread.sleep(1_000);
    }
    catch (InterruptedException ignored) { }
    return info + info;
}
```



Was ist daran unschön?

Komplexe Erzeugung von Messages (nur bei Bedarf)

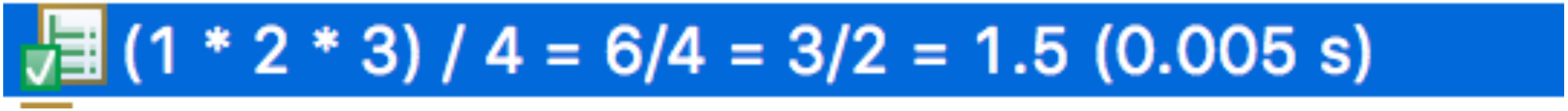
```java
@Test
void withMessageDelayed()
{
    String expected = "Tim";

    // complicated msg is only calculated if comparison fails
    assertEquals(expected, "Tim", () -> complicatedCalculation("Hint"));
    assertEquals(expected, "ALWAYS", () -> complicatedCalculation("Hint"));
}
```



Spezielle Testnamen

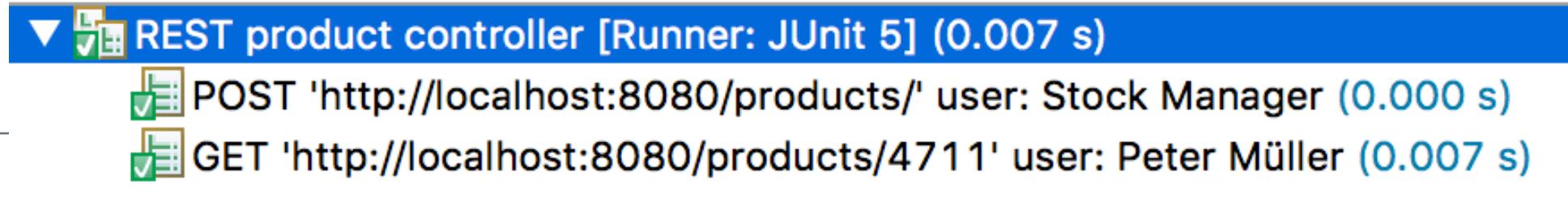
- Mit JUnit 4 war man auf „Java valide“ Methodennamen eingeschränkt
- Mit JUnit 5 fast beliebige, spezielle Testnamen mit der Annotation `@DisplayName` möglich

```
@Test
@DisplayName("(1 * 2 * 3) / 4 = 6/4 = 3/2 = 1.5")
void divideResultOfMultiplication()
{
    BigDecimal newValue = BigDecimal.ONE.multiply(BigDecimal.valueOf(2)).
                                         multiply(BigDecimal.valueOf(3)).
                                         divide(BigDecimal.valueOf(4));


    assertEquals(new BigDecimal("1.5"), newValue);
}
```



Spezielle Testnamen

```java
@DisplayName("REST product controller")
public class C_DisplayNameDemo
{
    @Test
    @DisplayName("GET 'http://localhost:8080/products/4711' user: Peter Müller")
    public void getProductFor4711()
    {
        // ..
    }

    @Test
    @DisplayName("POST 'http://localhost:8080/products/' user: Stock Manager")
    public void addProductAsStockManager()
    {
        // ...
    }
}
```



Spezielle Klassifikationen

- Spezielle Klassifikationen mit der Annotation `@Tag`

```
@Test
@DisplayName("(1 * 2 * 3) / 4 = 6/4 = 3/2 = 1.5")
@Tag("math")
void divideResultOfMultiplication()
{
    BigDecimal newValue = BigDecimal.ONE.multiply(BigDecimal.valueOf(2)).
                                         multiply(BigDecimal.valueOf(3)).
                                         divide(BigDecimal.valueOf(4));


    assertEquals(new BigDecimal("1.5"), newValue);
}
```

Kontextinfos `TestInfo`

- `TestInfo` dient als Ersatz für die Junit 4 Rule `TestName`
- **Dazu: Parameter in Testmethoden möglich!**

```
@Test
void simpleTestInfo(TestInfo ti)
{
    assertEquals("simpleTestInfo", ti.getTestMethod().get().getName());
}

@Test
@DisplayName("DEMO-TAGS")
@Tag("FAST")
@Tag("COOL")
void moreTestInfo(TestInfo ti)
{
    assertEquals("DEMO-TAGS", ti.getDisplayName());
    assertEquals(Set.of("FAST", "COOL"), ti.getTags());
}
```

Spezielle Assertions

Multilple Asserts

```java
@Test
void multipleAssertsforOneTopic()
{
    final Person mike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");

    // JUnit 4
    assertEquals("Mike", mike.name);
    assertEquals(LocalDate.of(1971, 2, 7), mike.dateOfBirth);
    assertEquals("Zürich", mike.homeTown);

    // JUnit 5
    assertAll(() -> assertEquals("Mike", mike.name),
              () -> assertEquals(LocalDate.of(1971, 2, 7), mike.dateOfBirth),
              () -> assertEquals("Zürich", mike.homeTown));
}
```

Wo liegt der Unterschied?

Multilple Asserts

```
@Test
void multipleAssertsforOneTopic_Diff1() {

    Person mike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");

    assertEquals("Tim", mike.name);
    assertEquals(LocalDate.of(1971, 3, 27), mike.dateOfBirth);
    assertEquals("Kiel", mike.homeTown);
}

@Test
void multipleAssertsforOneTopic_Diff2() {

    Person mike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");

    assertAll(() -> assertEquals("Tim", mike.name),
              () -> assertEquals(LocalDate.of(1971, 3, 27), mike.dateOfBirth),
              () -> assertEquals("Kiel", mike.homeTown));
}
```

Multilple Asserts

```java
@Test
void multipleAssertsforOneTopic_Diff1() {

    Person mike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");

    assertEquals("Tim", mike.name);
    assertEquals(LocalDate.of(1971, 3, 27), mike.dateOfBirth);
    assertEquals("Kiel", mike.homeTown);
}
```

```
org.opentest4j.AssertionFailedError: expected: <Tim> but was: <Mike>
    at a_first_slides.DisplayNameExample.multipleAssertsforOneTopic_Diff1(D
```

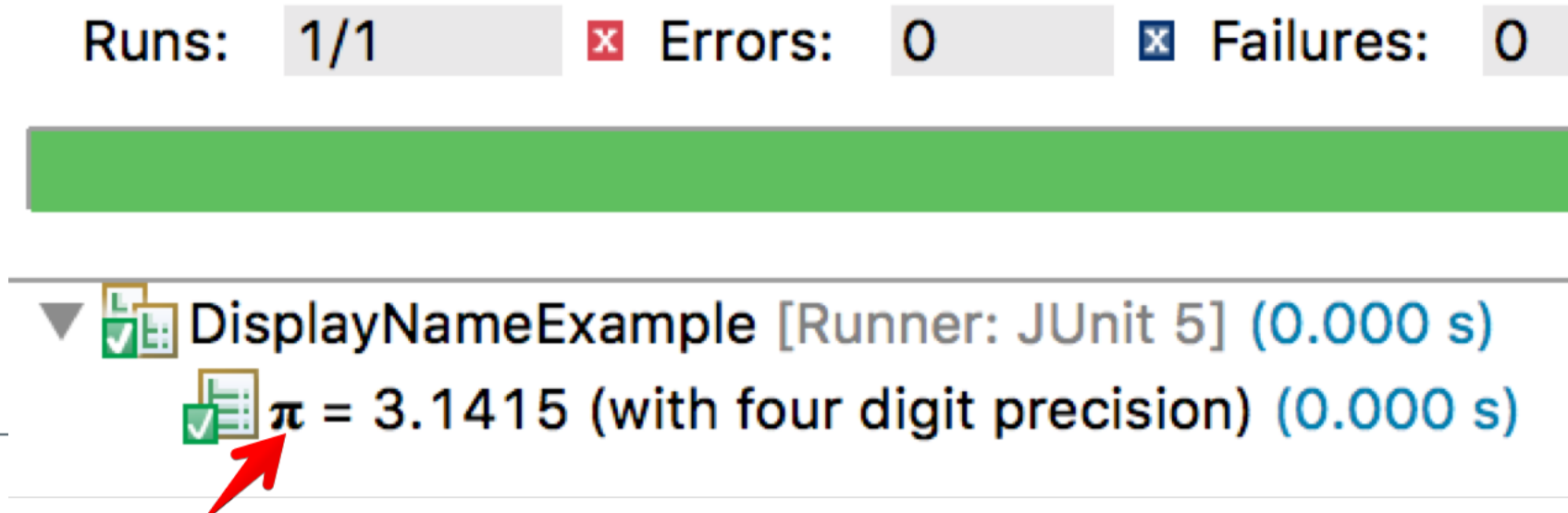
```java
@Test
void multipleAssertsforOneTopic_Diff2() {

    Person mike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");

    assertAll(() -> assertEquals("Tim", mike.name),
              () -> assertEquals(LocalDate.of(1971, 3, 27), mike.dateOfBirth),
              () -> assertEquals("Kiel", mike.homeTown));
}
```

```
org.opentest4j.MultipleFailuresError: Multiple Failures (3 failures)
expected: <Tim> but was: <Mike>
expected: <1971-03-27> but was: <1971-02-07>
expected: <Kiel> but was: <Zürich>
```

Wie testen wir Gleitkommazahlen?

Spezielles Handling bei Gleitkommazahlen

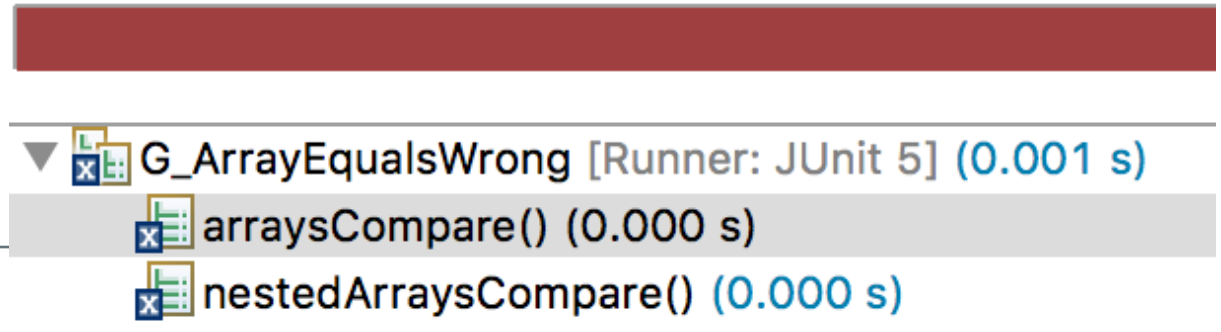
```java
@Test
@DisplayName("\uD835\uDED1 = 3.1415 (with four digit precision)")
void floatingArithemticRoundingForPI()
{
    double value = calculatePI();
    double precision = 0.0001;

    assertEquals(3.1415, value, precision);
}

private double calculatePI()
{
    return Math.PI;
}
```



Wie testen wir Arrays?

Arrays vergleichen – ein erster Versuch …

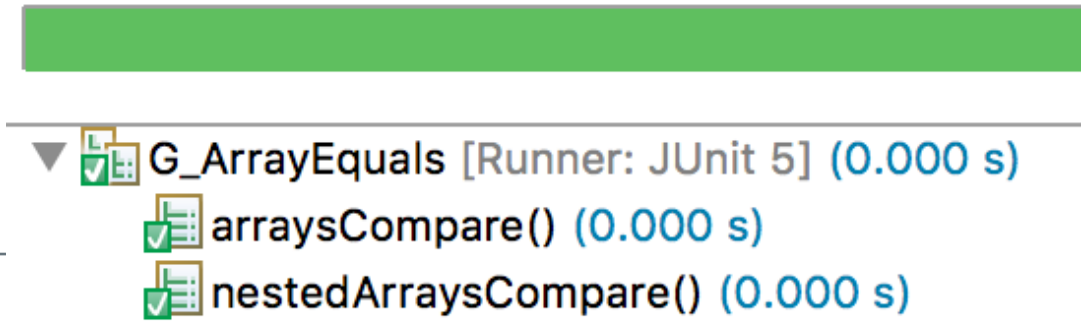
```java
@Test
void arraysCompare()
{
    final String[] words = { "Word1", "Word2" };
    final String[] expected = { "Word1", "Word2" };

    assertEquals(expected, words);
}

@Test
void nestedArraysCompare()
{
    final String[][] nested = { { "Line1", "Word1" },
                                { "Line2", "Word2" } };
    final String[][] expected = { { "Line1", "Word1" },
                                  { "Line2", "Word2" } };

    assertEquals(expected, nested);
}
```



Arrays RICHTIG vergleichen

ASMIQ

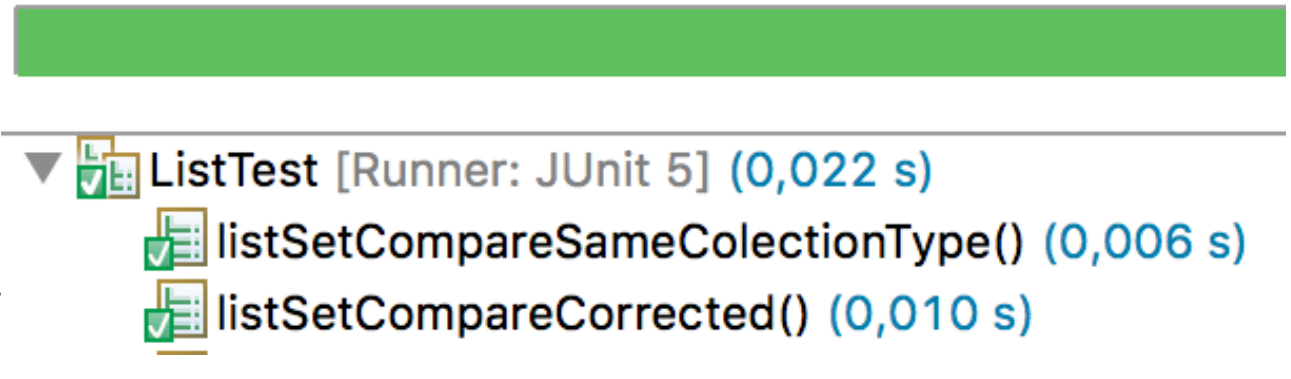
```java
@Test
void arraysCompare()
{
    final String[] words = { "Word1", "Word2" };
    final String[] expected = { "Word1", "Word2" };

    assertArrayEquals(expected, words);
}

@Test
void nestedArraysCompare()
{
    final String[][] nested = { { "Line1", "Word1" },
                                { "Line2", "Word2" } };
    final String[][] expected = { { "Line1", "Word1" },
                                  { "Line2", "Word2" } };

    assertArrayEquals(expected, nested);
}
```



Wie testen wir Collections?

Collections vergleichen --

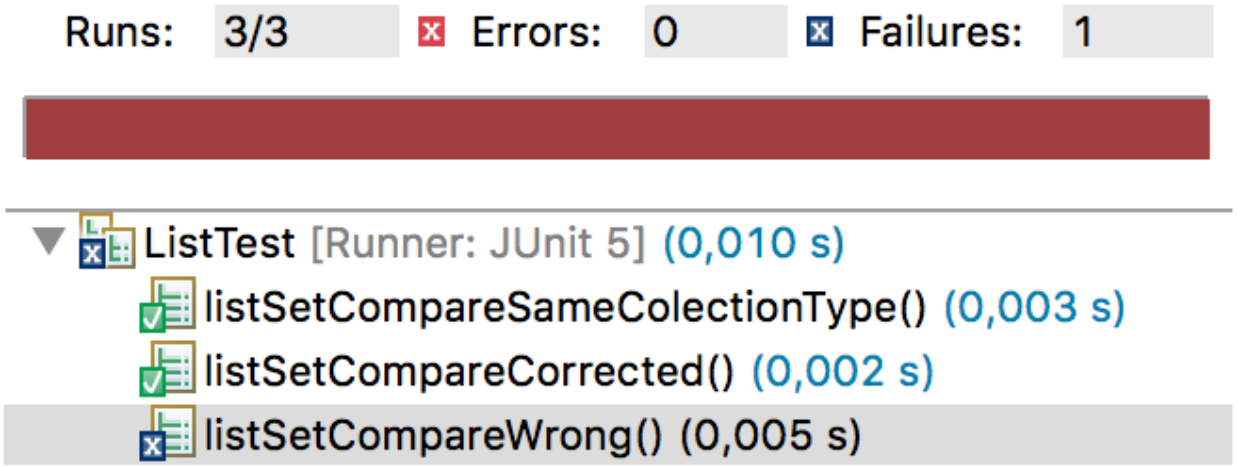
```java
@Test
void listSetCompareSameColectionType()
{
    final Collection<String> tags = new HashSet<>(Set.of("Fast", "Cool"));
    final Collection<String> names = List.of("Tim", "Mike", "Tom");

    assertEquals(Set.of("Fast", "Cool"), tags);
    assertEquals(List.of("Tim", "Mike", "Tom"), names);
}
```



Collections vergleichen --

```java
@Test
public void listSetCompareWrong()
{
    final List<String> actual = List.of("a", "b", "c", "d");
    final Set<String> expected = new TreeSet<>(Set.of("c", "a", "d", "b"));

    // Set und List vergleichen
    assertEquals(expected, actual);
}
```

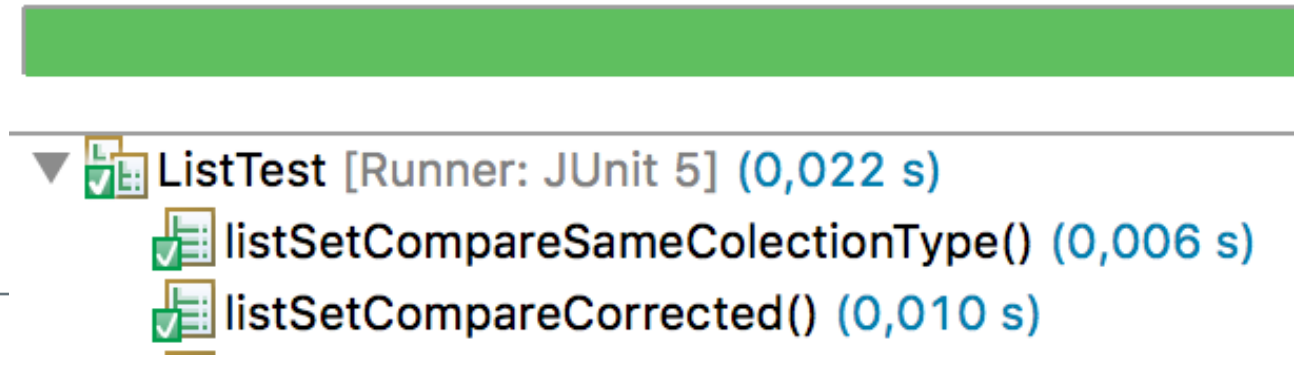
org.opentest4j.AssertionFailedError: expected: java.util.TreeSet@3b07a0d6<[a, b, c, d]> but was: java.util.ImmutableCollections$ListN@11a9e7c8<[a, b, c, d]>



Collections SICHERER vergleichen --

ASMIQ

```java
@Test
public void listSetCompareCorrected()
{
    final List<String> actual = List.of("a", "b", "c", "d");
    final Set<String> expected = new TreeSet<>(Set.of("c", "a", "d", "b"));

    // Set und List basierend auf Iterable vergleichen
    assertIterableEquals(expected, expected);
}
```



Testing Exceptions

Handarbeit

- Manchmal sollen Testfälle das Auftreten von Exceptions prüfen

```
try
{
    actionsThrowingAnException();
    fail();                              // Sollte hier nicht hinkommen
}
catch (final ExpectedException e)
{
    assertTrue(true);                    // Erwarteter Fall
}
```

Exceptions prüfen

- Seit JUnit 4 kann eine bei der Testausführung erwartete Exception in der Annotation `@Test` als Parameter `expected` angegeben werden:

```
@Test(expected = java.lang.NumberFormatException.class)
public void testFailWithExceptionJUnit4()
{
    // Hier wird bewusst ein Fehler provoziert
    final int value = Integer.parseInt("Fehler simulieren!");
    fail("calculation should throw an exception!");
}
```

- Problematisch: Wenn man den Message-Text auswerten möchte
- Auch kein AAA-Stil

JUnit Rules: Exceptions prüfen

- Besser JUnit Rule `ExpectedException`

```
@Rule
public ExpectedException thrown = ExpectedException.none();


@Test
public void testSomething()
{
   thrown.expect(IllegalStateException.class);
   thrown.expectMessage("XYZ is not initialized");

   doSomethingCausingAnExcpetion();
 }
```

Junit 5: assertThrows()

```
@Test
void cannotSetValueToNull()
{
    assertThrows(NullPointerException.class,
                 () -> new BigDecimal((String) null));
}
```

```
@Test
void assertThrowsException()
{
    assertThrows(IllegalArgumentException.class,
                 () -> { Integer.valueOf(null); });
}
```

JUnit 5: assertThrows() mit Rückgabe

```
@Test
void shouldThrowExceptionAndInspectMessage()
{
    Throwable exception = assertThrows(UnsupportedOperationException.class,
    () ->
    {
        throw new UnsupportedOperationException("Not supported");
    });

    assertEquals(exception.getMessage(), "Not supported");
}
```

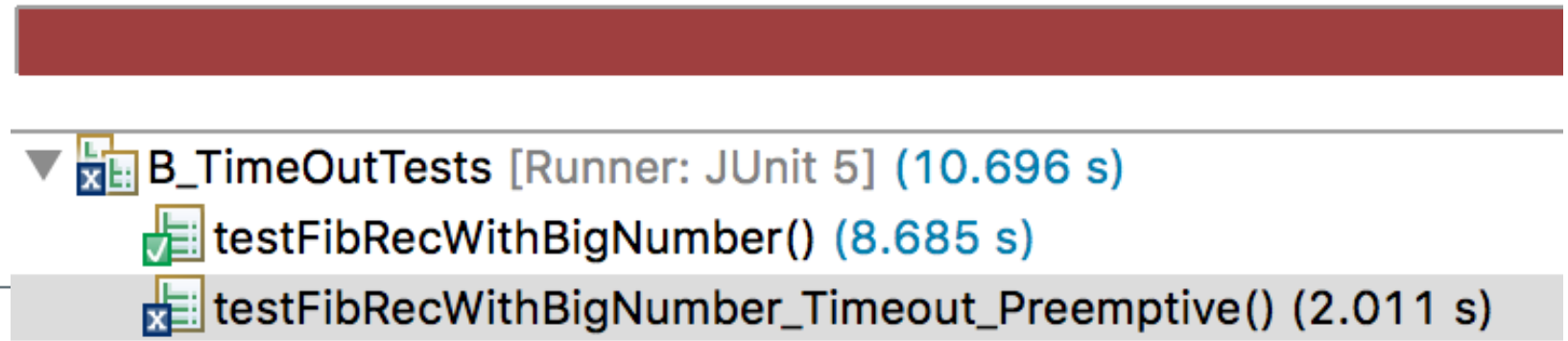
JUnit 5: assertThrows() mit Rückgabe

```
@Test
@DisplayName("Exception test clearer")
void exceptionTestImproved()
{
    Executable executable = () -> {
        throw new UnsupportedOperationException("Not supported");
    };

    Throwable exception = assertThrows(UnsupportedOperationException.class,
                                       executable);

    assertEquals(exception.getMessage(), "Not supported");
}
```

Timeout Assertions

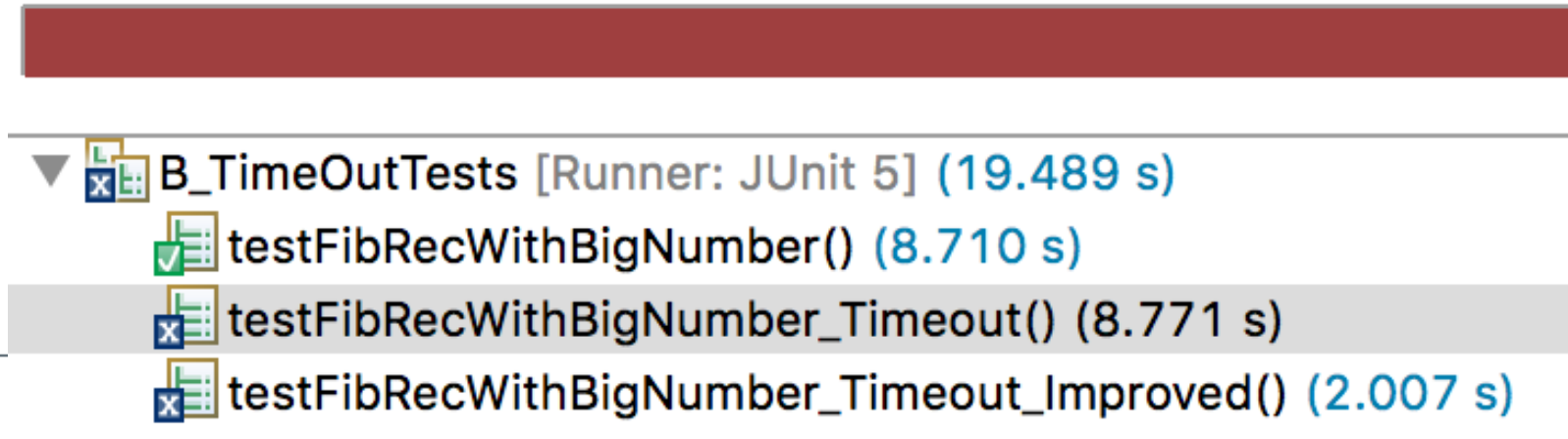
JUnit Rules: Time-outs prüfen

```
@Test
void testFibRecWithBigNumber()
{
    assertEquals(2971215073L, FibonacciCalculator.fibRec(47));
}

@Test
void testFibRecWithBigNumber_Timeout_Preemptive()
{
    assertTimeoutPreemptively(Duration.ofSeconds(2),
                              () -> FibonacciCalculator.fibRec(47));
}
```



JUnit Rules: Time-outs prüfen

```
@Test
void testFibRecWithBigNumber_Timeout()
{
    assertTimeout(Duration.ofSeconds(2),
                  () -> FibonacciCalculator.fibRec(47));
}

@Test
void testFibRecWithBigNumber_Timeout_Improved()
{
    assertTimeoutPreemptively(Duration.ofSeconds(2),
                              () -> FibonacciCalculator.fibRec(47));
}
```

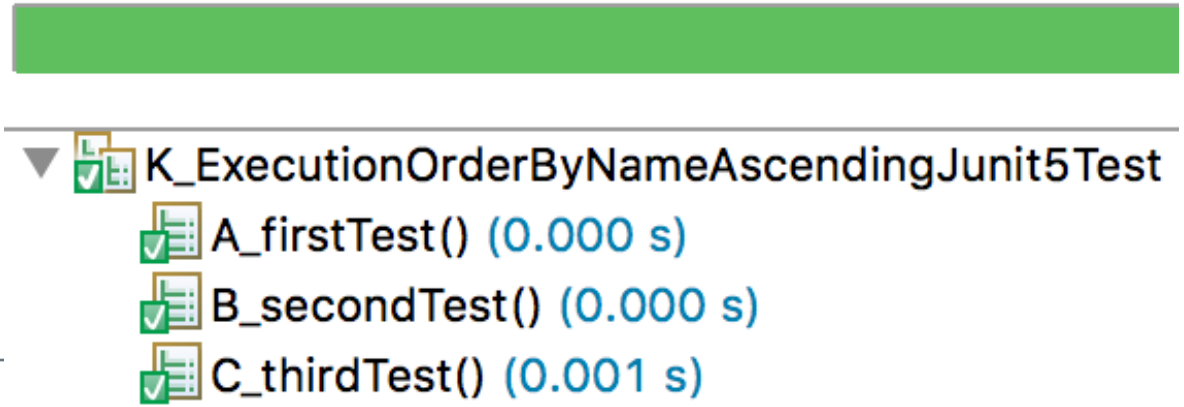


JUnit Test (temporär) ausschalten

```
@Test
@Disabled
void testFibRecWithBigNumber_Timeout()
{
    assertTimeout(Duration.ofSeconds(2),
                  () -> FibonacciCalculator.fibRec(47));
}
```

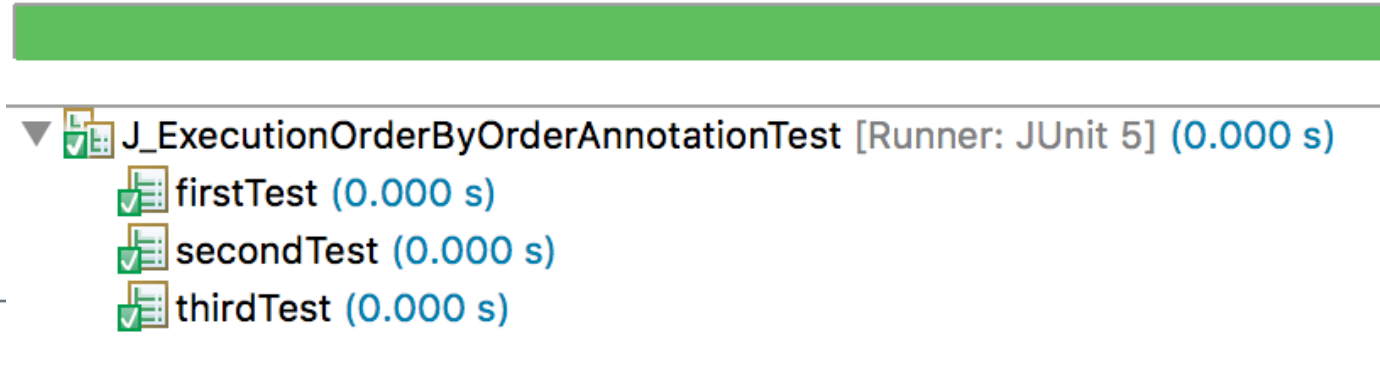
JUnit 5 Test Methoden ordnen (@TestMethodOrder)

```java
@TestMethodOrder(Alphanumeric.class)
public class K_ExecutionOrderByNameAscendingJunit5Test
{
    private final static StringBuilder result = new StringBuilder("");

    @Test
    public void B_secondTest() {
        result.append("b");
    }

    @Test
    public void C_thirdTest() {
        result.append("c");
    }

    @Test
    public void A_firstTest() {
        result.append("a");
    }

    @AfterAll
    public static void assertOutput()
    {
        assertEquals("abc", result.toString());
    }
}
```



JUnit 5 Test Methoden ordnen (@TestMethodOrder)

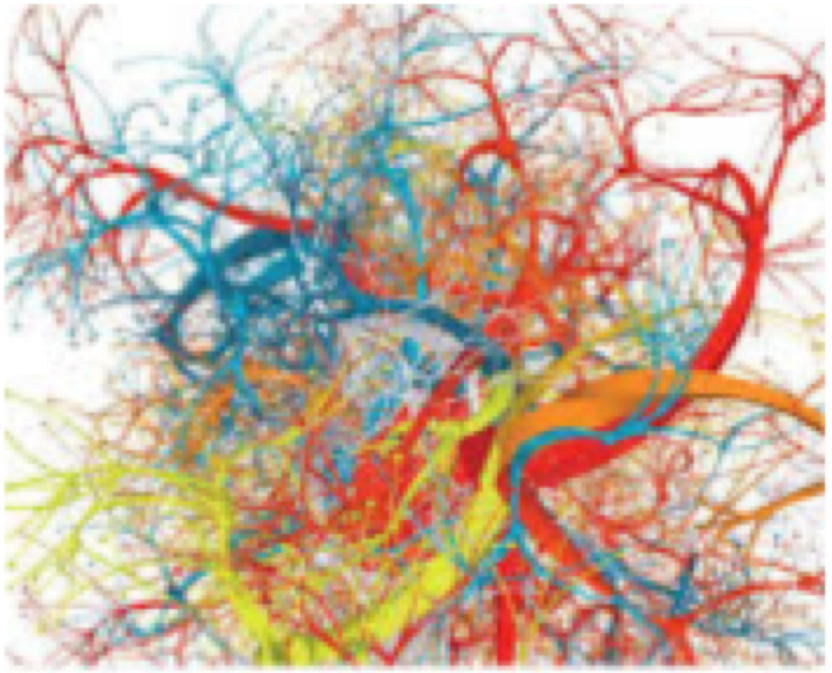
```
@TestMethodOrder(OrderAnnotation.class)
public class J_ExecutionOrderByOrderAnnotationTest
{
    private final static StringBuilder output = new StringBuilder("");

    @Test
    @Order(2)
    public void secondTest() {
        output.append("b");
    }

    @Test
    @Order(3)
    public void thirdTest() {
        output.append("c");
    }

    @Test
    @Order(1)
    public void firstTest() {
        output.append("a");
    }

    @AfterAll
    public static void assertOutput()
    {
        assertEquals("abc", output.toString());
    }
```



Exercises

Part 3
JUnit Advanced

Kombinatorik / Komplexität



3 zentrale Fragen

1. Welche Wertebelegungen soll man testen?

2. Wie vermeidet man zu viel Aufwand?

3. Wie finde ich diejenigen Testfälle, die eine gute und sichere Aussage über die Qualität und die Funktionalität ermöglichen?

Komplexität

- **Welche Wertebelegungen soll man testen?**
  - Selbst bei zwei `int` => $2^{32} * 2^{32} = 2^{64}$ Kombinationen
- **Wie vermeidet man zu viel Aufwand?**
  - Die wichtigen / komplexen Dinge testen
  - Keine Getter / Setter testen
  - Geschickte Wahl von Eingaben, so dass viele Varianten abgeprüft werden
- **Wie finde ich diejenigen Testfälle, die eine gute und sichere Aussage über die Qualität und die Funktionalität ermöglichen?**
  - **Äquivalenzklassentest**
  - **Grenzwerttest**

Äquivalenzklassen

- Gruppierung von Eingaben: Verschiedene Werte => gleiches Ergebnis
- Typisches Beispiel: Rabattberechnung

Wertebereich	Rabatt
count < 100	0 %
100 <= count <= 1000	4 %
count > 1000	7 %

- **Wie viele und welche Äquivalenzklassen ergeben sich?**

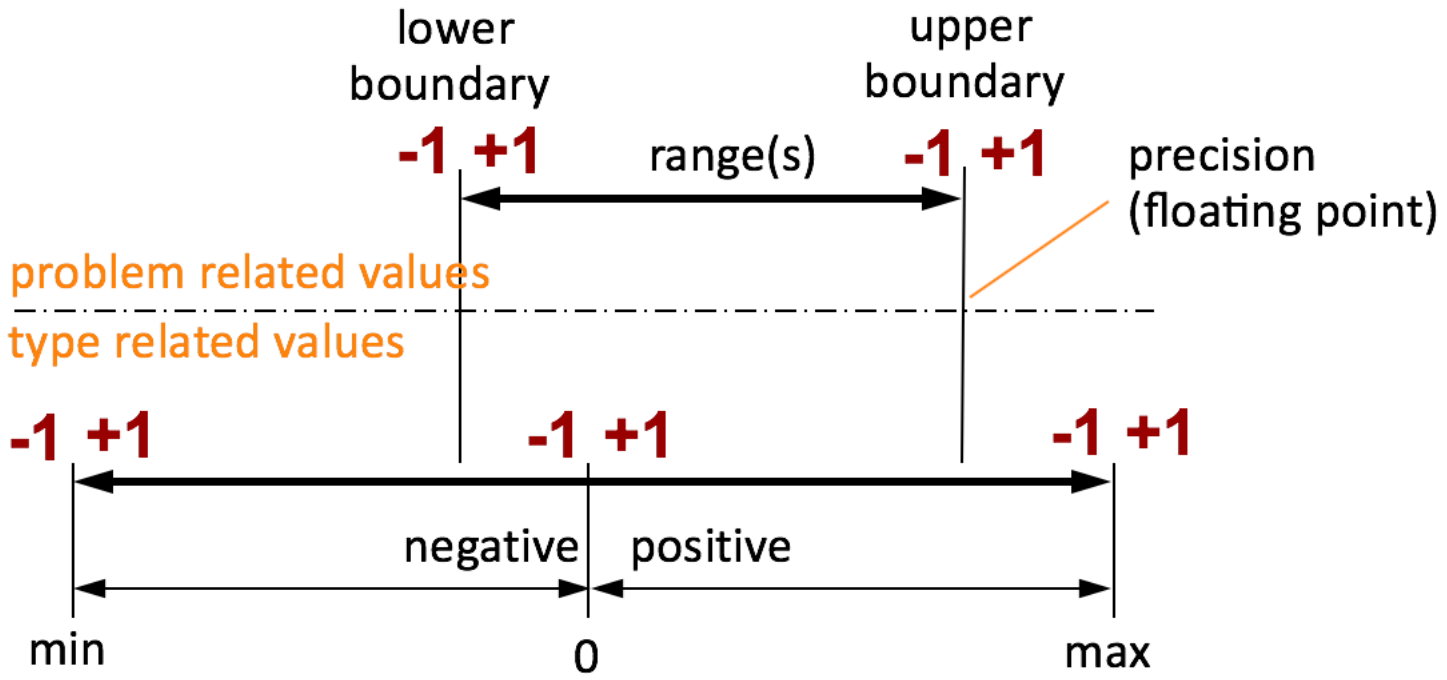
Äquivalenzklassentest

- **Schreiben wir also 3 Testmethoden. Aber: Reichen diese Tests aus?**

```
@Test
public void testCalcDiscount_SmallOrder_NoDiscount()
{
    final int smallAmount = 20;
    assertEquals(0, calculator.calcDiscount(smallAmount), "no discount");
}

@Test
public void testCalcDiscount_MediumOrder_MediumDiscount()
{
    final int mediumAmount = 200;
    assertEquals(4, calculator.calcDiscount(mediumAmount), "4 % discount");
}

@Test
public void testCalcDiscount_BigOrder_BigDiscount()
{
    final int bigAmount = 2000;
    assertEquals(7, calculator.calcDiscount(bigAmount), "7 % discount");
}
```

Grenzwerttests

- **NEIN!** Die Erfahrung aus der Praxis zeigt, man benötigt neben Äquivalenzklassentests noch weitere, warum?
- Oftmals finden wir **an den Rändern noch Probleme**, also im Übergang der Wertebereiche:



Grenzwerttests

- **Für die Rabattberechnung finden wir an den Rändern noch Probleme**, also im Übergang der Wertebereiche, hier also
  - 99, 100, 101
  - 999, 1000, 1001
- Wieso? Oftmals Flüchtigkeitsfehler bei Vergleichen mit < <= == != >= >, und +/-1
- Weitere potenzielle Kandidaten sind:
  - Werte < 0 oder
  - Werte > als ein vorgesehenes Maximum

Sollen wir etwa für alle diese Werte einzelne Methoden schreiben?

☹ ☹ ☹

Parameterized Tests

Parameterized Test

- **Abhilfe durch sogenannte Parameterized Test**
- **Testfall mit verschiedenen Daten immer wieder mit neuer Wertebelegung auszuführen**
- **Dadurch alle gewünschten, zu prüfenden Kombinationen abdecken**

- **Realisierungsvarianten**
  - Handarbeit: for-Schleife: liefert nur sukzessive Ergebnisse
  - JUnit 4 krampfig, syntaktisch unschön
  - JUnit 4 mit Expected Exception besser, aber wieder einiges an Eigenarbeit
  - JUnit 5 **endlich gut**

Parameterized Test: Kurzer Blick zurück: for-Schleife

ASMIQ

```
@Test
public void testCheckMatchingBracesAllOkay() throws Exception
{
	List<String> inputs = List.of("()", "()[]{}", "[((()[]{}))]");

	for (String current : inputs)
	{
		assertTrue("Checking " + current,
				MatchingBracesChecker.checkMatchingBraces(current));
	}
}

@Test
public void testCheckMatchingBracesAllWrong() throws Exception
{
	for (String current : List.of("(()", "(()}", "(()}", ")()("))
	{
		assertFalse("Checking " + current,
				MatchingBracesChecker.checkMatchingBraces(current));
	}
}
```

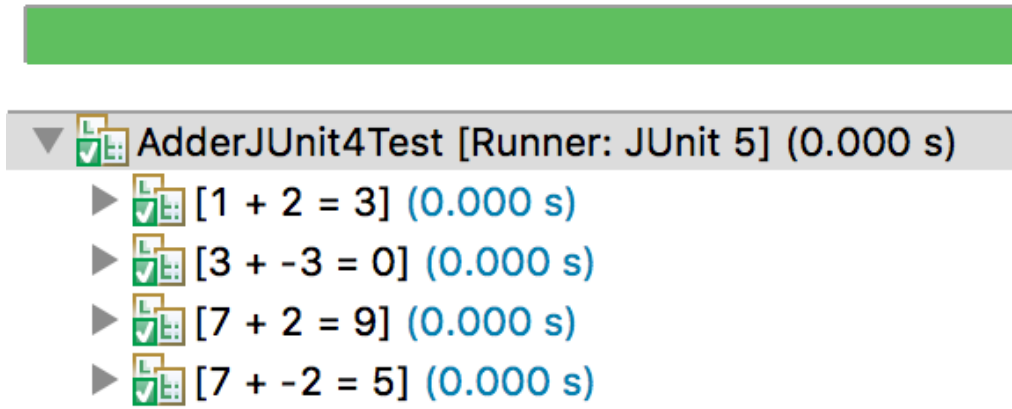
Parameterized Test: Kurzer Blick zurück: JUnit 4 Parameterized

- **Wie testen wir folgende Klasse Adder mit verschiedenen Wertekombinationen?**

```
public class Adder
{
    public int addNumbers(int a, int b)
    {
        return a + b;
    }
}
```

- **Schreiben wir also eine Testklasse mit**
  - *@RunWith(Parameterized.class)*
  - *Einem Konstruktor und allen Eingaben und Expected*
  - *Einer statischen Methode zum Generieren der Testdaten.*
  - *Einer Testmethode*

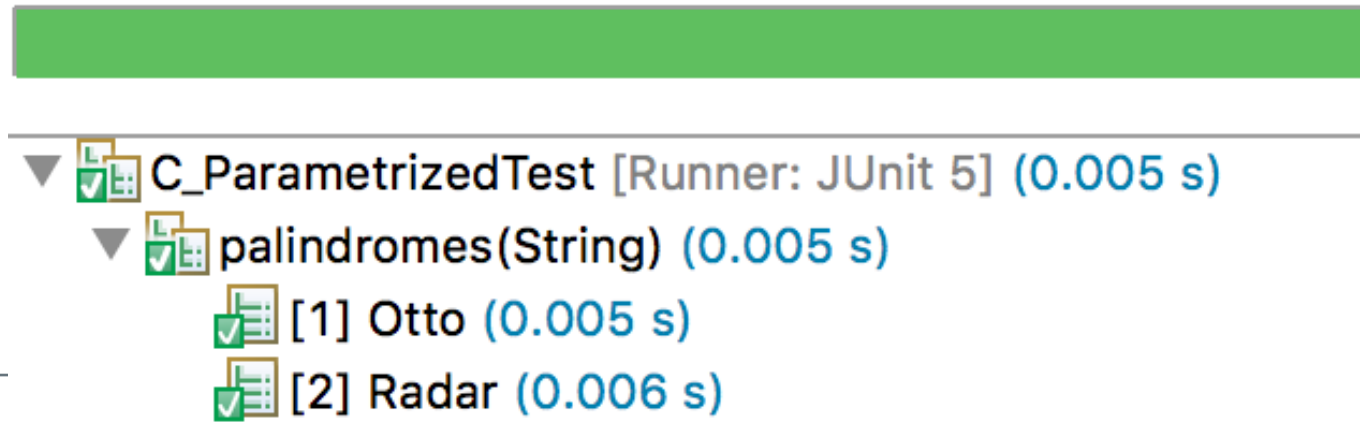
```java
@RunWith(Parameterized.class)
public class AdderJUnit4Test
{
    private int first;
    private int second;
    private int expected;

    public AdderJUnit4Test(int firstNumber, int secondNumber, int expectedResult)
    {
        this.first = firstNumber;
        this.second = secondNumber;
        this.expected = expectedResult;
    }

    @Parameters(name="{0} + {1} = {2}")
    public static Collection<Integer[]> inputAndExpectedNumbers()
    {
        return Arrays.asList(new Integer[][] { { 1, 2, 3 }, { 3, -3, 0 },
                                               { 7, 2, 9 }, { 7, -2, 5 } });
    }

    @Test
    public void sum()
    {
        assertEquals(expected, Adder.add(first, second));
    }
}
```



Das kann es ja jetzt nicht wirklich sein?!

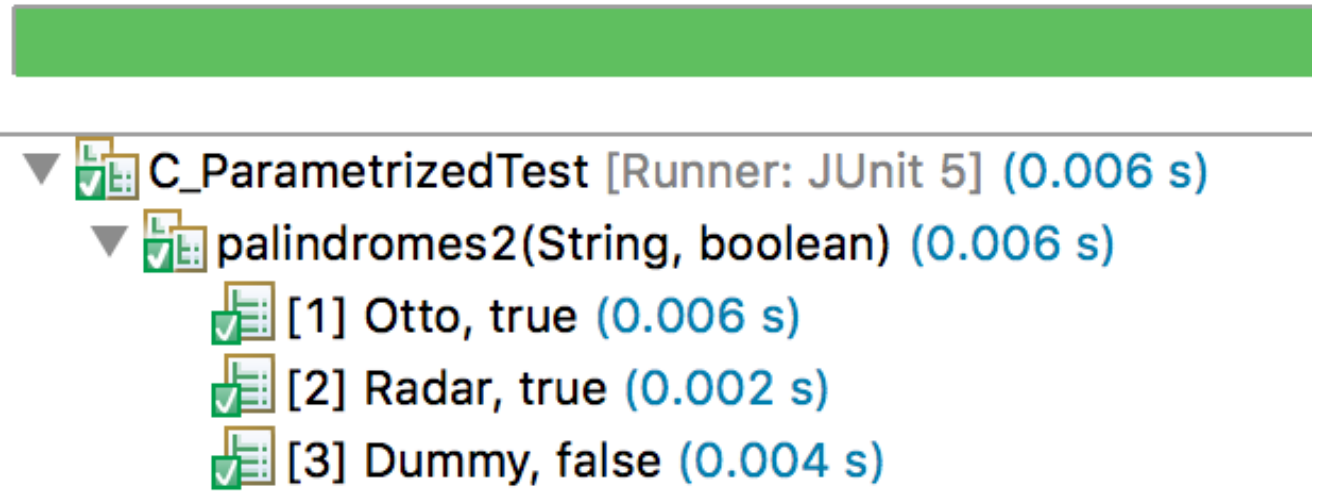
Parameterized Test – JUnit 5 @ParameterizedTest / @ValueSource

```
@ParameterizedTest
@ValueSource(strings = { "Otto", "Radar" })
void palindromes(String candidate)
{
    boolean isPalindrome = PalindromeChecker.isPalindrome(candidate));

    assertTrue(isPalindrome);
}
```



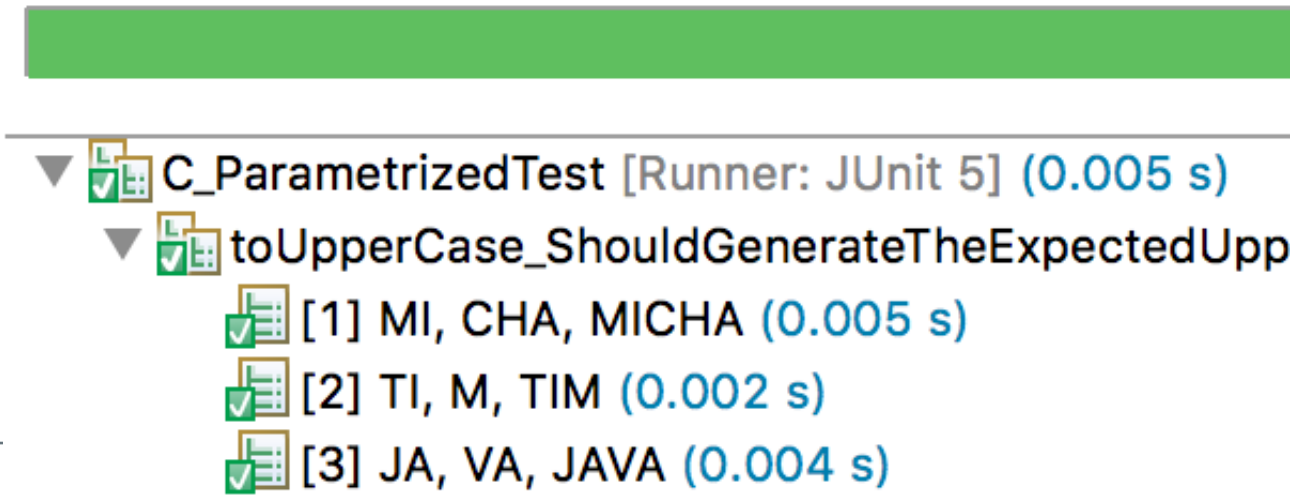
Parameterized Test – @CsvSource

```
@ParameterizedTest
@CsvSource({"Otto,true", "Radar,true", "Dummy,false" })
void palindromes2(String candidate, boolean expected)
{
    boolean isPalindrome = PalindromeChecker.isPalindrome(candidate));

    assertEquals(expected, isPalindrome);
}
```



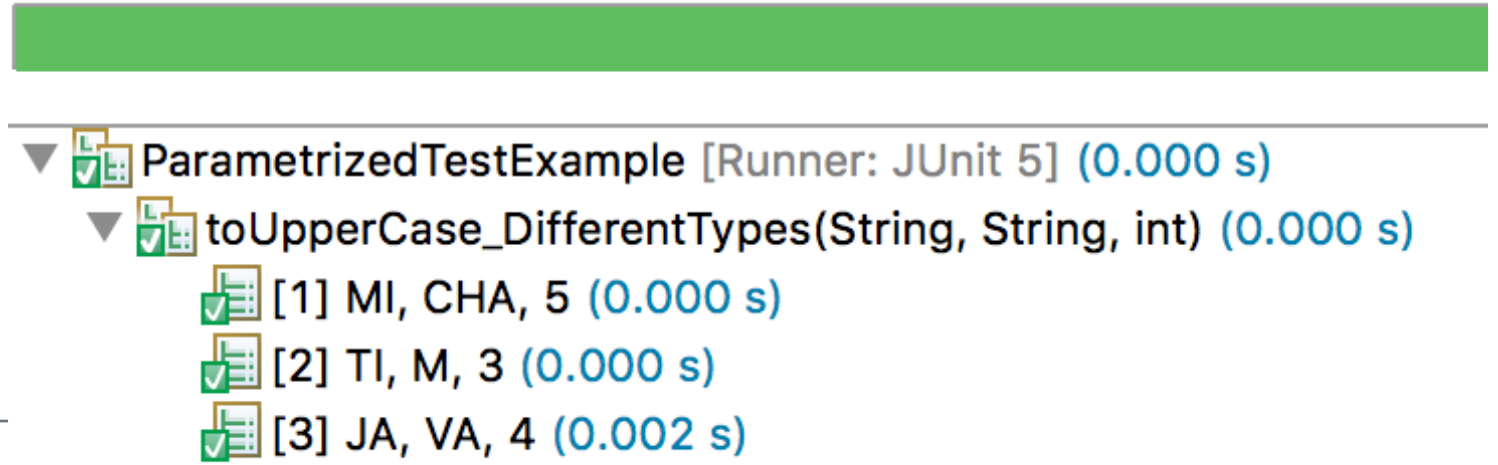
Parameterized Test – mehrere Eingaben

```
@ParameterizedTest
@CsvSource({"MI,CHA,MICHA", "TI,M,TIM", "JA,VA,JAVA"})
void toUpperCase_ShouldGenerateTheExpectedUppercaseValue(String input1,
                                                         String input2,
                                                         String expected)
{
    String actualValue = input1.concat(input2);

    assertEquals(expected, actualValue);
}
```



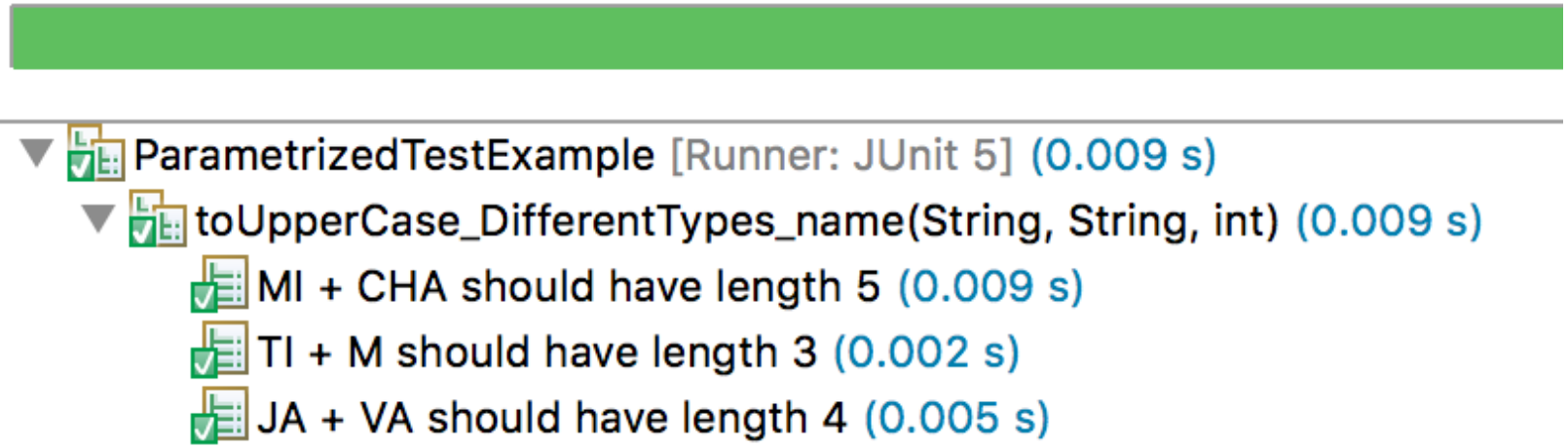
Parameterized Test – mehrere Eingaben verschiedene Typen

```
@ParameterizedTest
@CsvSource({"MI,CHA,5", "TI,M,3", "JA,VA,4"})
void toUpperCase_DifferentTypes(String input1,
                                String input2,
                                int expectedLength)
{
    int actualValue = input1.concat(input2).length();

    assertEquals(expectedLength, actualValue);
}
```



Parameterized Test – bessere Benennung

```
@ParameterizedTest(name = "{0} + {1} should have length {2}")
@CsvSource({"MI,CHA,5", "TI,M,3", "JA,VA,4"})
void toUpperCase_DifferentTypes_name(String input1,
                                     String input2,
                                     int expectedLength)
{
    int actualValue = input1.concat(input2).length();

    assertEquals(expectedLength, actualValue);
}
```

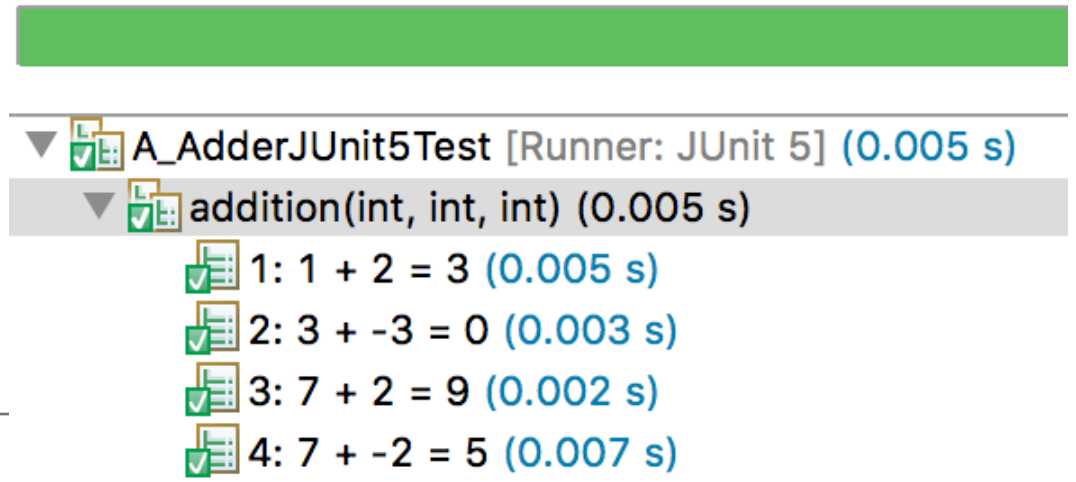


Wie sieht denn nun mit JUnit 5 der Test für den Adder aus?

Parameterized Test – Adder Test & bessere Benennung

ASMIQ

```
public class A_AdderJUnit5Test
{
    @ParameterizedTest(name = "{index}: {0} + {1} = {2}")
    @CsvSource({ "1,2,3", "3, -3, 0", "7, 2, 9", "7,-2,5" })
    void addition(int a, int b, int result)
    {
        int sum = Adder.add(a, b);

        assertEquals(result, sum);
    }
}
```

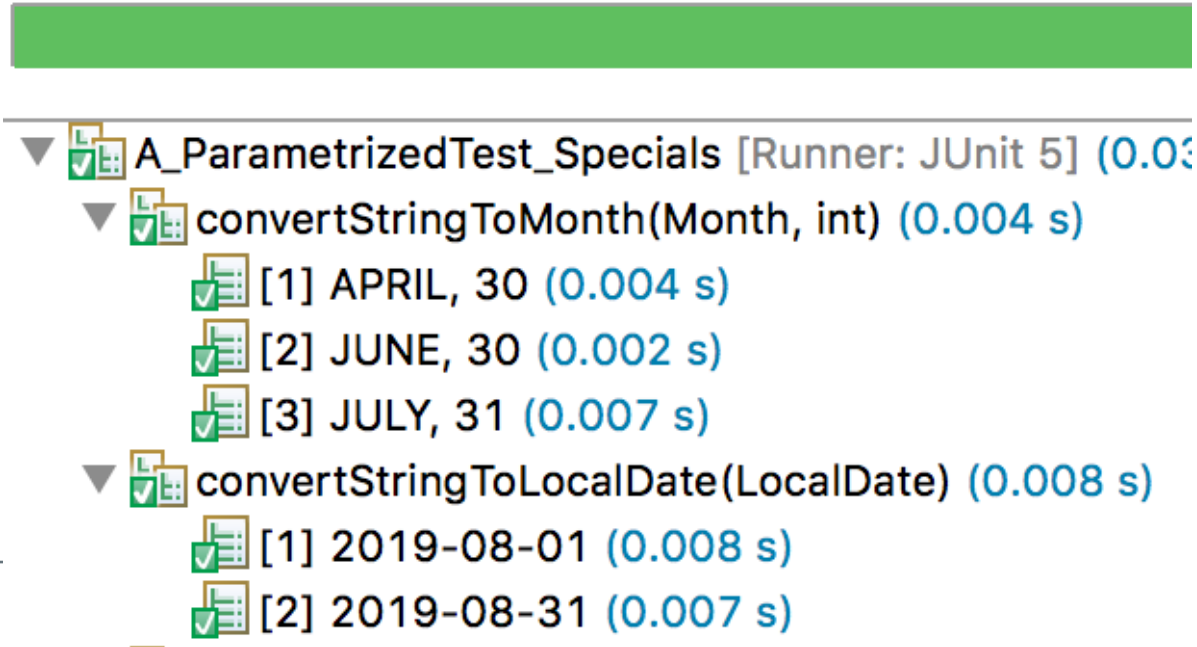


Parameterized Test – Diverse Konvertierungen und Hilfen

- *LocalDate, LocalTime, LocalDateTime, Year, Month, etc.*

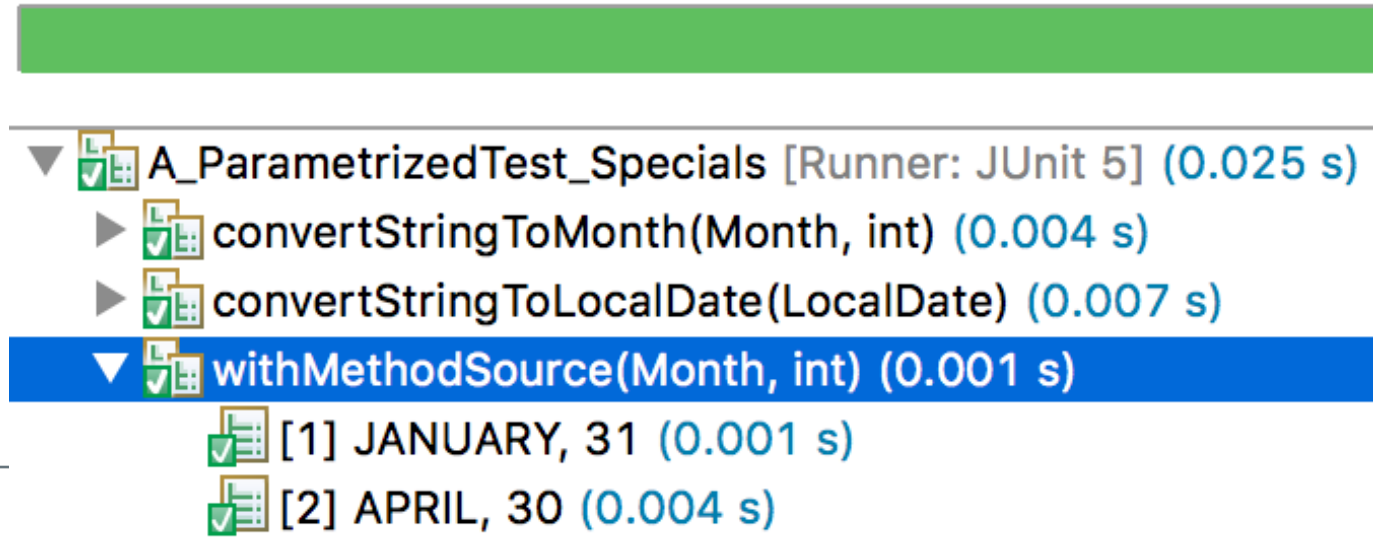
```
@ParameterizedTest
@ValueSource(strings = { "2019-08-01", "2019-08-31" })
void convertStringToLocalDate(LocalDate localDate)
{
    assertEquals(Month.AUGUST, localDate.getMonth());
}
```

```
@ParameterizedTest
@CsvSource(value= {"APRIL:30", "JUNE:30",
          "JULY:31"}, delimiter = ':')
void convertStringToMonth(Month month,
                          int length)
{
    assertEquals(length,
                 month.length(false));
}
```



Parameterized Test – @MethodSource, z. B. für Enums

```
@ParameterizedTest
@MethodSource("createMonthsWithLength")
void withMethodSource(Month month, int expectedLength)
{
    assertEquals(expectedLength, month.length(false));
}

private static Stream<Arguments> createMonthsWithLength()
{
    return Stream.of(Arguments.of(Month.JANUARY, 31),
                     Arguments.of(Month.APRIL, 30));
}
```



Was lässt sich mit Parameterized Test denn noch so machen?

Parameterized Test – @MethodSource, z. B. für Listen als Parameter

ASMIQ

```
@ParameterizedTest(name = "removeDuplicates({0}) = {1}")
@MethodSource("listInputsAndExpected")
void removeDuplicates(List<Integer> inputs, List<Integer> expected)
{
    List<Integer> result = Ex02_ListRemove.removeDuplicates(inputs);


    assertEquals(expected, result);
}


static Stream<Arguments> listInputsAndExpected()
{
    return Stream.of(Arguments.of(List.of(1, 1, 2, 3, 4, 1, 2, 3),
                                  List.of(1, 2, 3, 4)),
                     Arguments.of(List.of(1, 3, 5, 7),
                                  List.of(1, 3, 5, 7)),
                     Arguments.of(List.of(1, 1, 1, 1),
                                  List.of(1)));
}
```

Parameterized Test – @CsvSource-Besonderheiten

ASMIQ

```
@ParameterizedTest(name="adjustToPayday({0}) => {1}, {2}")
@CsvSource({"2019-07-21, 2019-07-25, normale Anpassung",
            "2019-06-27, 2019-07-25, normale Anpassung auf nächsten Monat",
            "2019-08-21, 2019-08-23, Freitag; falls 25. am Wochenende",
            "2019-12-06, 2019-12-16, Dezember: Mitte Monat und Montag " +
                                     "nach Wochenende",
            "2019-12-23, 2020-01-24, nächster Monat und Freitag; falls 25. " +
                                     "am Wochenende"})
public void adjustInto(LocalDate startDay, LocalDate expected, String info)
{
    final TemporalAdjuster paydayAdjuster = new Ex12_NextPaydayAdjuster();

    final Temporal result = paydayAdjuster.adjustInto(startDay);

    assertEquals(expected, result);
}
```

Parameterized Test – @CsvSource, z. B. für grosse Datenmengen

ASMIQ

```
@ParameterizedTest(name = "fromRomanNumber(''{1}'') => {0}")
@CsvSource({ "1, I", "2, II", "3, III", "4, IV", "5, V", "7, VII", "9, IX",
             "17, XVII", "40, XL", "90, XC", "400, CD", "444, CDXLIV", "500, D",
             "900, CM", "1000, M", "1666, MDCLXVI", "1971, MCMLXXI",
             "2018, MMXVIII", "2019, MMXIX", "2020, MMXX", "3000, MMM"})
@DisplayName("Konvertiere römische in arabische Zahl")
void fromRomanNumber(final int arabicNumber, final String romanNumber)
{
    int result = Ex07_RomanNumbers.fromRomanNumber(romanNumber);

    assertEquals(arabicNumber, result);
}
```

Parameterized Test – @CsvSource, z. B. für grosse Datenmengen

ASMIQ

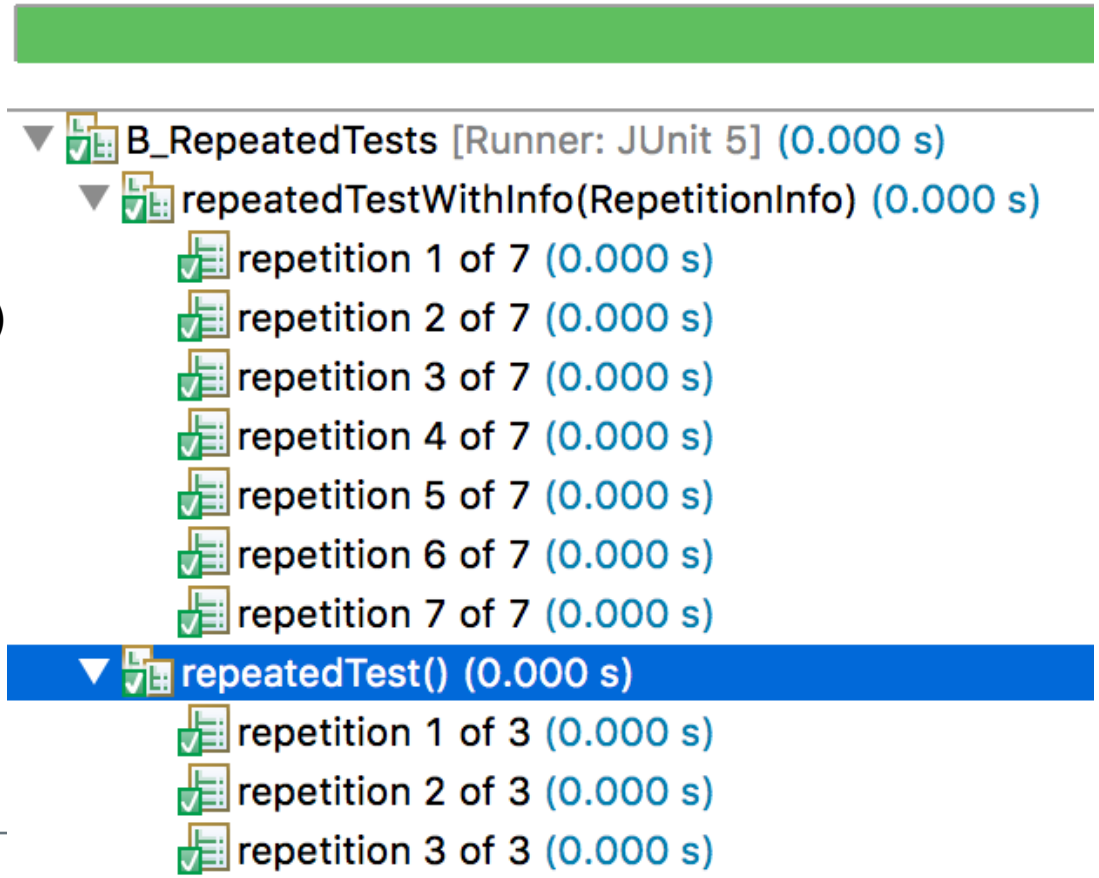
```
@ParameterizedTest(name = "fromRomanNumber(''{1}'') => {0}")
@CsvFileSource(resources = "arabicroman.csv", numLinesToSkip = 1)
@DisplayName("Konvertiere römische in arabische Zahl")
void fromRomanNumber(final int arabicNumber, final String romanNumber)
{
    int result = Ex07_RomanNumbers.fromRomanNumber(romanNumber);

    assertEquals(arabicNumber, result);
}
```

```
arabic,roman
1, I
2, II
3, III
4, IV
5, V,
7, VII
9, IX
17, XVII
40, XL
90, XC
```

Wie kann man Tests mehrmals ausführen und warum sollte man das wollen?

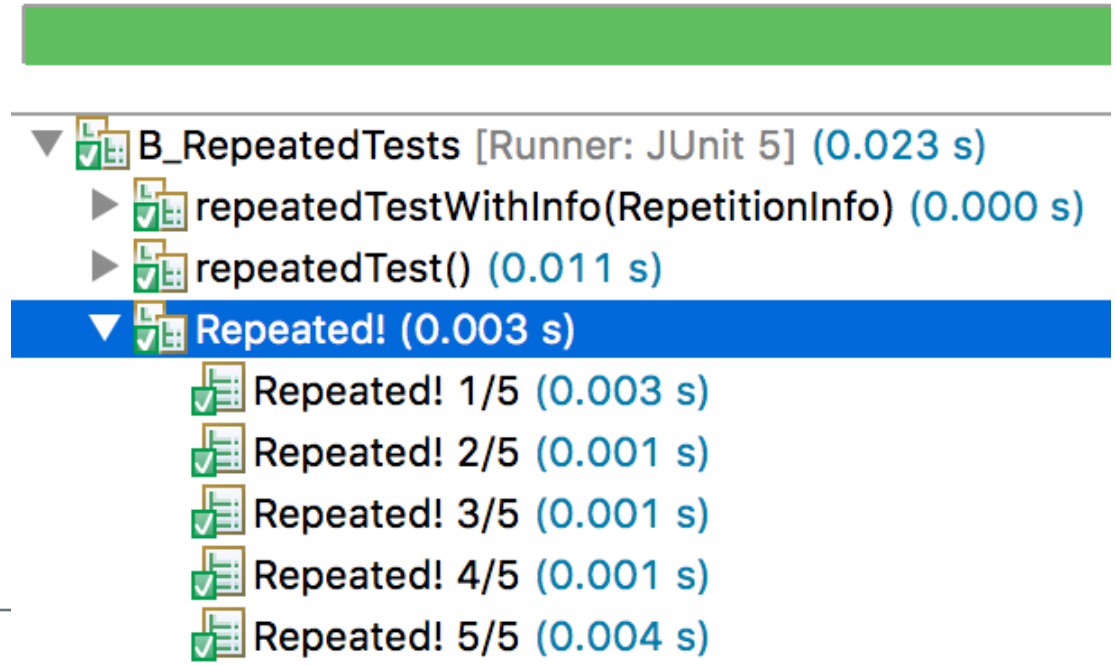
Repeated Tests

```java
@RepeatedTest(3)
void repeatedTest()
{
    assertTrue(true);
}

@RepeatedTest(7)
void repeatedTestWithInfo(RepetitionInfo info)
{
    assertEquals(7, info.getTotalRepetitions())
}
```



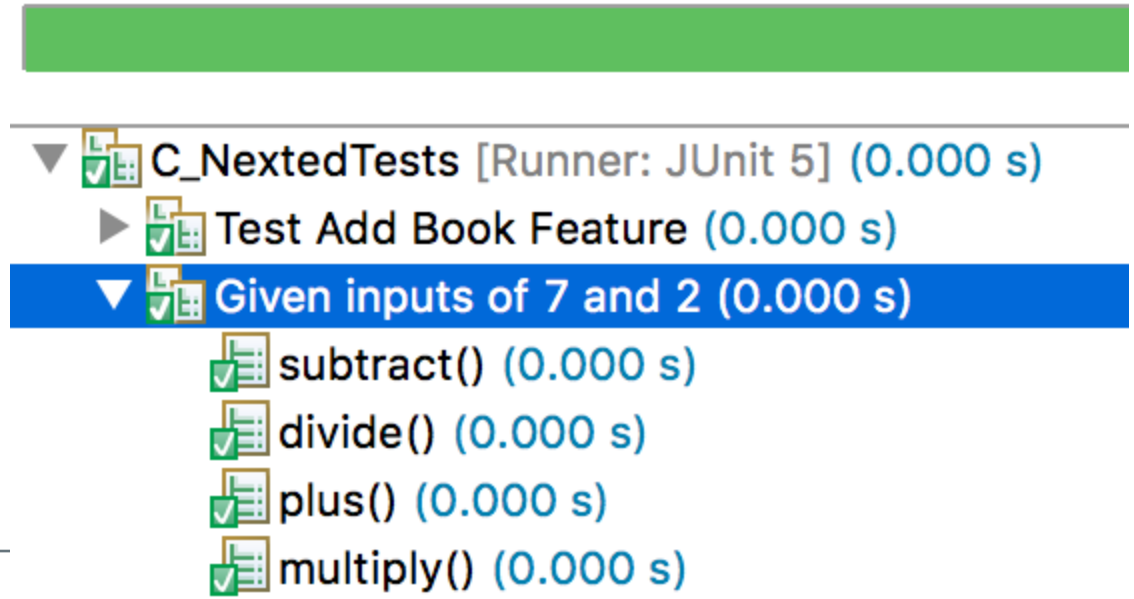
Repeated Tests

```
@RepeatedTest(value = 5,
              name = "{displayName} {currentRepetition}/{totalRepetitions}")
@DisplayName("Repeated! ")
void customDisplayName(TestInfo testInfo, RepetitionInfo repetitionInfo)
{
    int current = repetitionInfo.getCurrentRepetition();
    int total = repetitionInfo.getTotalRepetitions();

    assertEquals(testInfo.getDisplayName(),
                 "Repeated! " + current +
                 "/" + total);
}
```



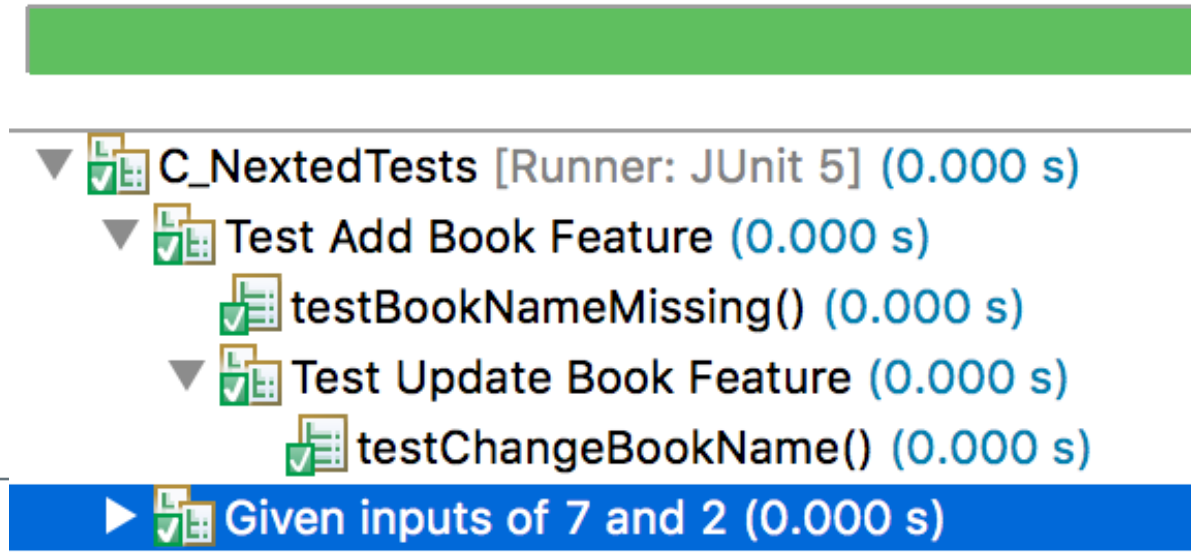
Nested Tests

```
@Nested
@DisplayName("Given inputs of 7 and 2")
class PredefinedValues
{
    final int input1 = 7;
    final int input2 = 2;

    @Test
    void plus()
    {
        assertEquals(9, input1 + input2);
    }

    @Test
    void subtract()
    {
        assertEquals(5, input1 - input2);
    }

    // …
}
```

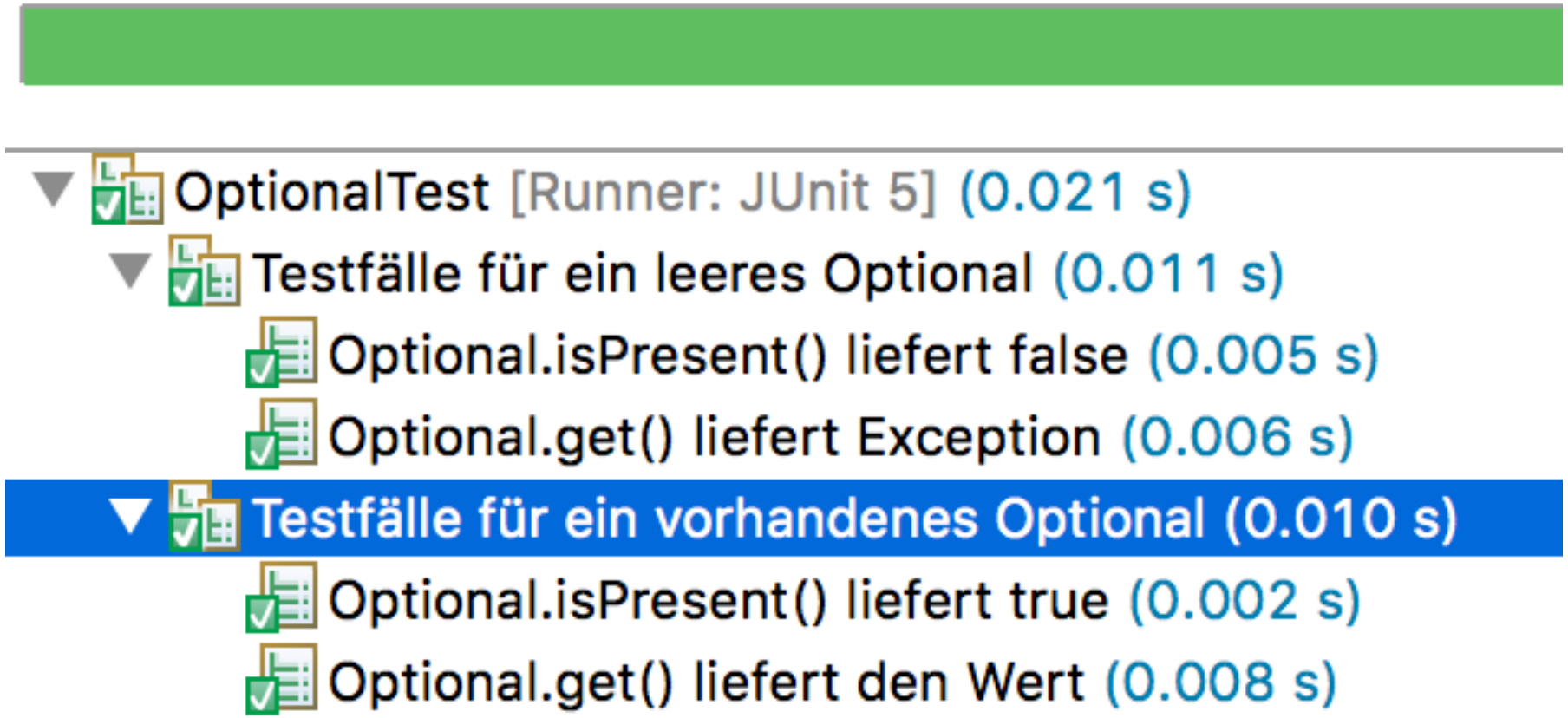


Nested Tests

```
@Nested
@DisplayName("Test Add Book Feature")
class AddFeature
{
    @Test
    void testBookNameMissing()
    {
    }

    @Nested
    @DisplayName("Test Update Book Feature")
    class UpdateFeature
    {
        @Test
        void testChangeBookName()
        {
        }
    }
}
```



Nested Tests



https://www.viadee.de/wp-content/uploads/JUnit5_javaspektrum.pdf

Simple Extensions

Simple Extensions: Benchmarking

```
public class BenchmarkExtension implements BeforeTestExecutionCallback,
                                           AfterTestExecutionCallback
{
    private long start;

    @Override
    public void beforeTestExecution(ExtensionContext ctx) throws Exception
    {
        start = System.currentTimeMillis();
    }

    @Override
    public void afterTestExecution(ExtensionContext ctx) throws Exception
    {
        System.err.println("Test " + ctx.getDisplayName() + " took " +
                           (System.currentTimeMillis() - start) + " ms");
    }
}
```

Simple Extensions: Benchmarking

```
@ExtendWith(BenchmarkExtension.class)
public class BenchmarkedFibonacciTest
{
    @Test
    void testFibRecWithBigNumber()
    {
        long value = FibonacciCalculator.fibRec(47);

        assertEquals(2971215073L, value);
    }

    @Test
    void testFibRecWithBigNumber_Timeout()
    {
        assertTimeoutPreemptively(Duration.ofSeconds(2),
                                  () -> FibonacciCalculator.fibRec(47));
    }
}
```

```
Test testFibRecWithBigNumber() took 8675 ms
Test testFibRecWithBigNumber_Timeout() took 2012 ms
```

PART 4:
Tipps zur Migration

Wissenswertes zum direkten Starten

- Vermutlich habt ihr schon eine grössere Basis an Tests => **Migrationsplan** nötig
- **Migration** oder / und **Parallelbetrieb** möglich: **Parallelbetrieb** bietet sich an
- Praktisch: Zwar ähnliche Annotations aber in anderen Packages
- JUnit 4 parallel zu JUnit 5:

```
// JUnit 4 Support
testCompile "junit:junit:4.12"
testRuntime "org.junit.vintage:junit-vintage-engine:5.5.1"
```

- Einige JUnit 4 Rules nutzen:

```
// Migration Support to Enable Rules in JUnit 5
testCompile "org.junit.jupiter:junit-jupiter-migrationsupport:5.5.1"
```

Wissenswertes zum direkten Starten

- Beide: Testfälle in Form spezieller Testmethoden, die mit `@Test` markiert sein müssen.

- JUnit 4:
  - Testmethoden `public`
  - *Keine* Parameter erlaubt
  - Package: `org.junit`
  - Parameterreihenfolge: `assertTrue("Always true", true);`
  - `assertThat()` und einige Hamcrest-Matcher inkludiert

- JUnit 5:
  - Testmethoden nicht mehr zwingend `public`
  - Kann Parameter haben
  - Package: `org.junit.jupiter.api`
  - Parameterreihenfolge: `assertTrue(true, "Always true");`
  - *Kein* `assertThat()` und *keine* Hamcrest-Matcher

Annotations JUnit 4 vs JUnit 5

JUnit 4	JUnit 5		Beschreibung
`org.junit`	`org.junit.jupiter.api`		Package
`@Test`	`@Test`		Definiert einen Testfall
`@Ignore`	`@Disabled`		Testfall (temporär) deaktivieren
`@BeforeClass`	`@BeforeAll`		Aktion ***einmalig*** vor allen Testmethoden
`@Before`	`@BeforeEach`		Aktion ***jeweils*** vor allen Testmethoden
`@After`	`@AfterEach`		Aktion ***jeweils*** nach allen Testmethoden
`@AfterClass`	`@AfterAll`		Aktion ***einmalig*** nach allen Testmethoden
`@Rule`	`-/-`		Erweiterungen, in JUnit 5 mit speziellen Methoden

Umstellung auf JUnit 5

- Auf lange Sicht ist eine Migration / Umstellung auf JUnit 5 ratsam
- Schrittweise Klasse für Klasse, Package für Package
  - Imports löschen und auf Package: `org.junit.jupiter.api` anpassen
  - Annotations ggf. leicht anpassen @BeforeXXX, @AfterXXX
  - Parameterreihenfolge beachten
  - `@Rule ExpectedException` durch JUnit-5-Features ersetzen, z. B. `assertThrows()`
  - `@Rule TimeOut` durch JUnit-5-Features ersetzen, z. B. `assertTimeout()`

Umstellung auf JUnit 5

- Schrittweise Klasse für Klasse, Package für Package
  - @EnableRuleMigrationSupport aus "org.junit.jupiter:junit-jupiter-migrationsupport:5.5.1" einbinden
    - TemporaryFolder
    - ErrorCollector
    - ExpectedException
  - `assertThat()` durch `AssertJ` oder `Hamcrest` ersetzen

Was bringt mir AssertJ?

AssertJ – https://assertj.github.io/doc/

- JAR in die IDE aufnehmen / Build-Datei anpassen

```
testCompile group: 'org.assertj', name: 'assertj-core', version: '3.13.2'
```

- **import static** org.assertj.core.api.Assertions.*;

```
@Test
void assertJAssertinsBasics()
{
    String peter = "Peter";
    assertThat(peter).isEqualTo("Peter");

    assertThat(peter.isEmpty()).isFalse();
    assertThat("".isEmpty()).isTrue();

    assertThat(7.271).isEqualTo(7.2, withPrecision(0.1d));
}
```

AssertJ – Collections

ASMIQ

```
@Test
void assertJCollectionsBasics()
{
    List<String> names = List.of("Tim", "Tom", "Mike");

    assertThat(names).isNotEmpty();
    assertThat(names).contains("Mike");
    assertThat(names).startsWith("Tim");

    assertAll(()-> assertThat(names).isNotEmpty(),
              ()-> assertThat(names).contains("Mike"),
              ()-> assertThat(names).startsWith("Tim"));

    // AssertJ Variante Chaining
    assertThat(names).isNotEmpty().
                      contains("Mike").
                      startsWith("Tim");
}
```

AssertJ – Maps

```
@Test
void assertJMapsBasics()
{
    Map<String, Integer> personsAgeMap = Map.of("Tim", 48, "Tom", 7, "Mike", 48);


    assertThat(personsAgeMap).isNotEmpty()
                             .containsKey("Mike")
                             .doesNotContainKeys("Peter")
                             .contains(Map.entry("Tim", 48));
}
```

Tipp:

```
assertThat(personList).contains(mike).isSortedAccordingTo(byAge);
```

Exercises

PART 5:
Test Smells

Test Smells High Level

- **Ratschlag: Folge deiner Nase ☺**

- Prinzipiell sind Tests auch nur Sourcecode wie Businesscode
- Test Smells sind ähnlich zu Bad Smells / Code Smells (Details im “Java-Profi“)
- Bad Smell Examples: https://www.youtube.com/watch?v=9E6_zpx3q2c

- Man findet darüber hinaus folgende Test Smells auf High Level
    - Tests sind schwierig zu schreiben
    - Merkwürdige, komplizierte Dinge nötig, um Tests zu schreiben / zum Laufen zu bringen
    - Es benötigt sehr viel Mocking
    - Testen erfordert manchmal sogar Spezialbehandlungen im Businesscode
    - Die Testausführung ist (zu) langsam
    - Die Testausführung liefert schwankende Resultate (Random Failures)

Test Smell		Symptom
Hard-to-Test Code		Code is difficult to test
Fragile Test		A test fails to compile or run when the SUT is changed in ways that do not affect the part the test is exercising.
Erratic Test		One or more tests are behaving erratically; sometimes they pass and sometimes they fail.
Obscure Test		It is difficult to understand the test at a glance
Assertion Roulette		It is hard to tell which of several assertions within the same test method caused a test failure.
Slow Tests		The tests take too long to run.
Test Code Duplication		The same test code is repeated many times **=> Gilt NICHT für einfache Initialisierungen.**
Test Logic in Production		The code that is put into production contains logic that should be exercised only during tests. **=> Gilt NICHT für getter!* Anekdote Reflection !!!**

http://xunitpatterns.com/Test%20Smells.html

Assertion Roulette / Obscure Test – Not SRP – Multiple Test Cases

ASMIQ

```java
@Test
public void testFlightMileage_asKm2() throws Exception
{
    // setup fixture
    String validFlightNumber = "LX 857";
    // exercise contructor
    Flight newFlight = new Flight(validFlightNumber);
    // verify constructed object
    assertEquals(validFlightNumber, newFlight.number);
    assertEquals("LX", newFlight.airlineCode);
    // setup mileage
    newFlight.setMileage(1111);
    int actualKilometres = newFlight.getMileageAsKm();
    // verify results
    int expectedKilometres = 1777;
    assertEquals(expectedKilometres, actualKilometres);
    // now try it with a canceled flight:
    newFlight.cancel();
    Throwable th = assertThrows(InvalidRequestException.class,
                                () -> newFlight.getMileageAsKm());
    assertEquals("Cannot get cancelled flight mileage", th.getMessage());
}
```

(Basierend auf http://xunitpatterns.com/Assertion%20Roulette.html)

Smell: Falsche Nutzung von `assertTrue()/False()`

ASMIQ

```
assertTrue(db.writeCount == 10);
assertTrue(tasks.totalProcessed == 10);
assertTrue(tasks.errorCount == 0);
```

Smell: Falsche Nutzung von assertTrue()/False()

```
assertTrue(db.writeCount == 10);
assertTrue(tasks.totalProcessed == 10);
assertTrue(tasks.errorCount == 0);
```

```
assertEquals(10, db.writeCount);
assertEquals(10, tasks.totalProcessed);
assertEquals(0, tasks.errorCount);
```

Einmal hin alles drin ☹ assertTrue() forever? ☹

ASMIQ

```java
@Test
void assertTrueForever()
{
    final Person mike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");
    final Person sameMike = mike;
    final Person otherMike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");

    assertTrue(mike != null, "mike not null");
    assertTrue(mike == sameMike, "same obj");
    assertTrue(mike.equals(otherMike), "same content");
}
```

Einmal hin alles drin ☹ assertTrue() forever? ☹

ASMIQ

```java
@Test
void assertTrueForever()
{
    final Person mike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");
    final Person sameMike = mike;
    final Person otherMike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");

    assertTrue(mike != null, "mike not null");
    assertTrue(mike == sameMike, "same obj");
    assertTrue(mike.equals(otherMike), "same content");
}
```

```java
@Test
void rightAssertsForTheJob()
{
    final Person mike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");
    final Person sameMike = mike;
    final Person otherMike = new Person("Mike", LocalDate.of(1971, 2, 7), "Zürich");

    assertNotNull(mike, "mike not null");
    assertSame(mike, sameMike, "same obj");
    assertEquals(mike, otherMike, "same content");
}
```

Smell: Zu viele Asserts

ASMIQ

```
assertEquals(10, db.readCount);
assertEquals(10, db.writeCount);
assertEquals(10, db.commitCount);

assertEquals(10, tasks.totalProcessed);
assertEquals(0, tasks.errorCount);
assertEquals(SUCCESS, tasks.status);
```

Smell: Einsatz von `toString()` in `assertEquals()`

ASMIQ

```
assertEquals("mongodb.writeConcern.timeout=10000," +
             "mongodb.writeConcern.writes=1, " +
             "mongodb.port=27017, " +
             "mongodb.password=ksdjsa2455aAYdsj," +
             "mongodb.user=ABCD",
             dbConnection.toString())
```

Questions?

ASMIQ


HELP
IS ON THE
WAY


4.
Auflage
Michael Inden
Der Weg zum
Java-Profi
Konzepte und Techniken für die
professionelle Java-Entwicklung
Aktuell
zu
Java 9
dpunkt.verlag


Java
Michael Inden
Challenge
Fit für das Job-Interview und die Praxis –
mit mehr als 100 Aufgaben
und Musterlösungen
dpunkt.verlag


Michael Inden
Java
Die Neuerungen in
Version 9 bis 14
Modularisierung, Syntax- und
API-Erweiterungen
dpunkt.verlag

Weitere Infos / Quellen

ASMIQ

- **JUnit 5**
  - https://junit.org/junit5/
  - https://jaxenter.de/highlights-junit-5-65986
  - https://jaxenter.de/junit-5-beyond-testing-framework-81787
  - https://gul.gu.se/public/pp/public_courses/course82759/published/1524658283418/resourceId/40520654/content/junit-tdd-mocking.pdf
  - https://www.viadee.de/wp-content/uploads/JUnit5_javaspektrum.pdf

- **AssertJ**
  - https://assertj.github.io/doc/
  - https://joel-costigliola.github.io/assertj/assertj-core-quick-start.html
  - https://www.vogella.com/tutorials/AssertJ/article.html
  - https://dzone.com/articles/assertj-and-collections-introduction
  - https://de.slideshare.net/tsveronese/assert-j-techtalk (Hamcrest vs. AssertJ)

Thank You