



JUnit 5

Ein Unit-Testing-Framework für Java

Gliederung

- Begriffe
- Aufbau von JUnit 5
- Tests Schreiben
- Annotations und Assertions
- Code-Beispiele

Was ist ein Unit?

- Logisch trennbarer Teil eines Computer-Programms
- In Java: Methode, Klasse, Package, usw.

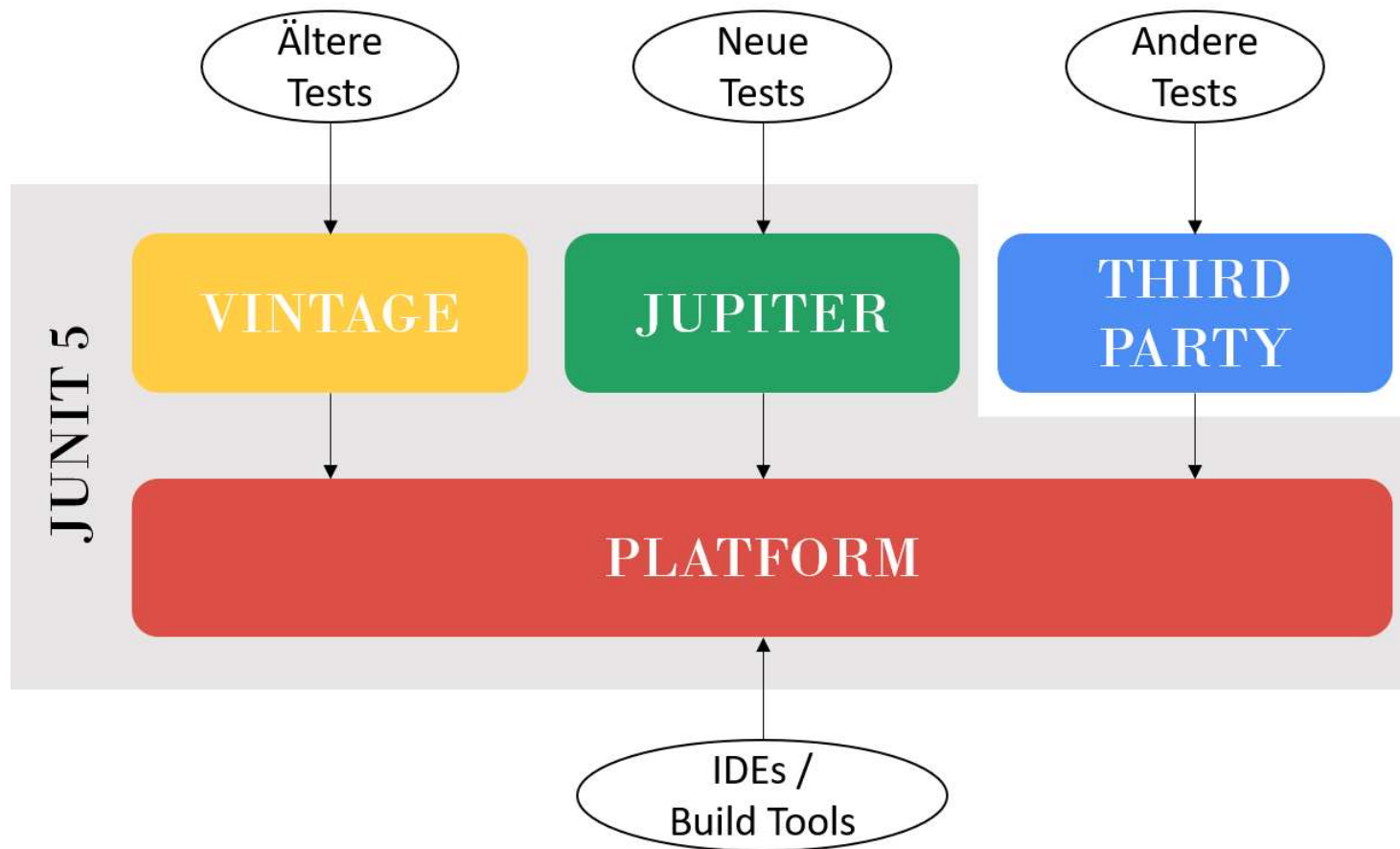
Was ist Unit-Testing?

- Eine Testtechnik
- Testen der Korrektheit der Funktionalität eines Units
- Testen von spezifiziertem Verhalten eines Units
- Units isolieren und Fehler analysieren
- Fehlerbehebung und erneut Testen

Aufbau von JUnit 5:

- JUnit Platform – Zur Ausführung von Tests auf der JVM
- JUnit Jupiter – Der neuste Test-Standard (Stand 2021)
- JUnit Vintage – Grundlage für Backwards-Compatibility mit JUnit 3 und 4

Aufbau von JUnit 5:



Nutzung von JUnit:

- Typischerweise mit Build Tools und IDEs (bspw. IntelliJ mit Gradle)
- JUnit Platform auch durch Kommandozeile ausführbar
- In dieser Präsentation: IntelliJ mit Maven und JUnit 5.7.2

Tests in JUnit:

- Annotations definieren und steuern Tests (z.B. @Test, @RepeatedTest)
- Assertions überprüfen die Ergebnisse der Tests/Testfälle

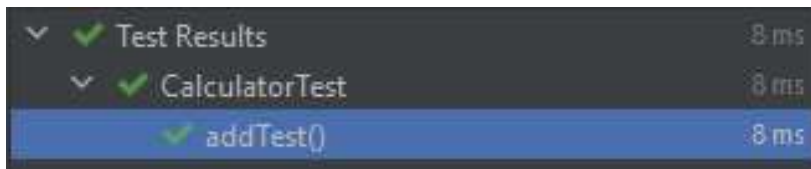
Allgemeiner Aufbau von Unit-Tests:

- Testszenario Aufbauen
- Test Ausführen
- Testergebnis überprüfen

Tests in JUnit:

- Separate Test-Klasse für zu testende Klasse erstellen
- Testen von allen zu testenden Methoden
- Beispiel:

```
public class Calculator {  
    public int add(int a, int b) {  
        return a + b;  
    }  
}
```



| | | |
|---|----------------|------|
| ✓ | Test Results | 8 ms |
| ✓ | CalculatorTest | 8 ms |
| ✓ | addTest() | 8 ms |

```
import static org.junit.jupiter.api.Assertions.*;  
import org.junit.jupiter.api.*;  
  
class CalculatorTest {  
    // Szenario aufbauen  
    1 {  
        int a = 1;  
        int b = 1;  
        private final Calculator calculator = new Calculator();  
    }  
  
    @Test  
    public void addTest() {  
        // Methode aufrufen und tatsächliches Ergebnis speichern  
        2 {  
            int actualResult = calculator.add(a, b);  
        }  
        // Überprüfung mit einer JUnit Assertion  
        3 {  
            assertEquals(2, actualResult);  
        }  
    }  
}
```


Wichtige und relevante Annotations:

- `@Test`, `@RepeatedTest(<Number>)`, `@TestFactory`, `@ParameterizedTest`
(Definieren Tests)
- `@BeforeEach`, `@AfterEach`, `@BeforeAll`, `@AfterAll`
(Dienen zum Szenario Aufbauen/Abbauen)

Wichtigste Assertion:

- `assertEquals(Object o1, Object o2)`
- Prüft auf Gleichheit zwischen zwei Objekten
- Wenn eine Assertion fehlschlägt, schlägt der Test fehl

@BeforeEach, @AfterEach, @BeforeAll, @AfterAll

```
public class Calculator {  
  
    double ans = Double.NaN;  
    double memory = Double.NaN;  
  
    public void memorize(){  
        memory = ans;  
    }  
    public void forget(){  
        this.memory = Double.NaN;  
    }  
  
    public double add(double a, double b) {...}  
    public double mul(double a, double b){...}  
    public double sub(double a, double b){...}  
    public double div(double a, double b) throws Exception {...}  
    public double log2(double a) {...}  
    public double pow(double a, double b){...}  
    public int fib(int n) {...}  
    public int fastFib(int n){...}  
  
}
```

Einmal vor
allen Tests

Einmal nach
allen Tests

Vor jedem Test

Nach jedem Test

```
class CalculatorTest {  
  
    private static Calculator calculator;  
  
    @BeforeAll  
    public static void start(){  
        calculator = new Calculator();  
        System.out.println("started testing...");  
    }  
  
    @AfterAll  
    public static void finish(){  
        System.out.println("ended testing...");  
    }  
  
    @BeforeEach  
    public void preparation(){  
        System.out.println("forgot: " + calculator.memory);  
        calculator.forget();  
    }  
  
    @AfterEach  
    public void finalization(){  
        calculator.memorize();  
        System.out.println("remembered: " + calculator.memory);  
    }  
  
    ...  
  
}
```

@RepeatedTest(<Number>)

```
public class Calculator {  
  
    double ans = Double.NaN;  
    double memory = Double.NaN;  
  
    public void memorize(){  
        memory = ans;  
    }  
    public void forget(){  
        this.memory = Double.NaN;  
    }  
  
    public double add(double a, double b) {...}  
    public double mul(double a, double b){...}  
    public double sub(double a, double b){...}  
    public double div(double a, double b) throws Exception {...}  
    public double log2(double a) {...}  
    public double pow(double a, double b){...}  
    public int fib(int n) {...}  
    public int fastFib(int n){...}  
  
}
```

```
class CalculatorTest {  
    ...  
  
    @RepeatedTest(5)  
    public void addTest() {  
        assertEquals(2, calculator.add(1,1));  
        assertEquals(2, calculator.ans);  
    }  
  
    ...  
}
```

| | |
|-------------------|-------|
| Test Results | 24 ms |
| CalculatorTest | 24 ms |
| addTest() | 24 ms |
| repetition 1 of 5 | 24 ms |
| repetition 2 of 5 | |
| repetition 3 of 5 | |
| repetition 4 of 5 | |
| repetition 5 of 5 | |

@ParameterizedTest

```
public class Calculator {  
  
    double ans = Double.NaN;  
    double memory = Double.NaN;  
  
    public void memorize(){  
        memory = ans;  
    }  
    public void forget(){  
        this.memory = Double.NaN;  
    }  
  
    public double add(double a, double b) {...}  
    public double mul(double a, double b){...}  
    public double sub(double a, double b){...}  
    public double div(double a, double b) throws Exception {...}  
    public double log2(double a) {...}  
    public double pow(double a, double b){...}  
    public int fib(int n) {...}  
    public int fastFib(int n){...}  
}
```

Parametrisierter Test

Stellt die Parameter bereit

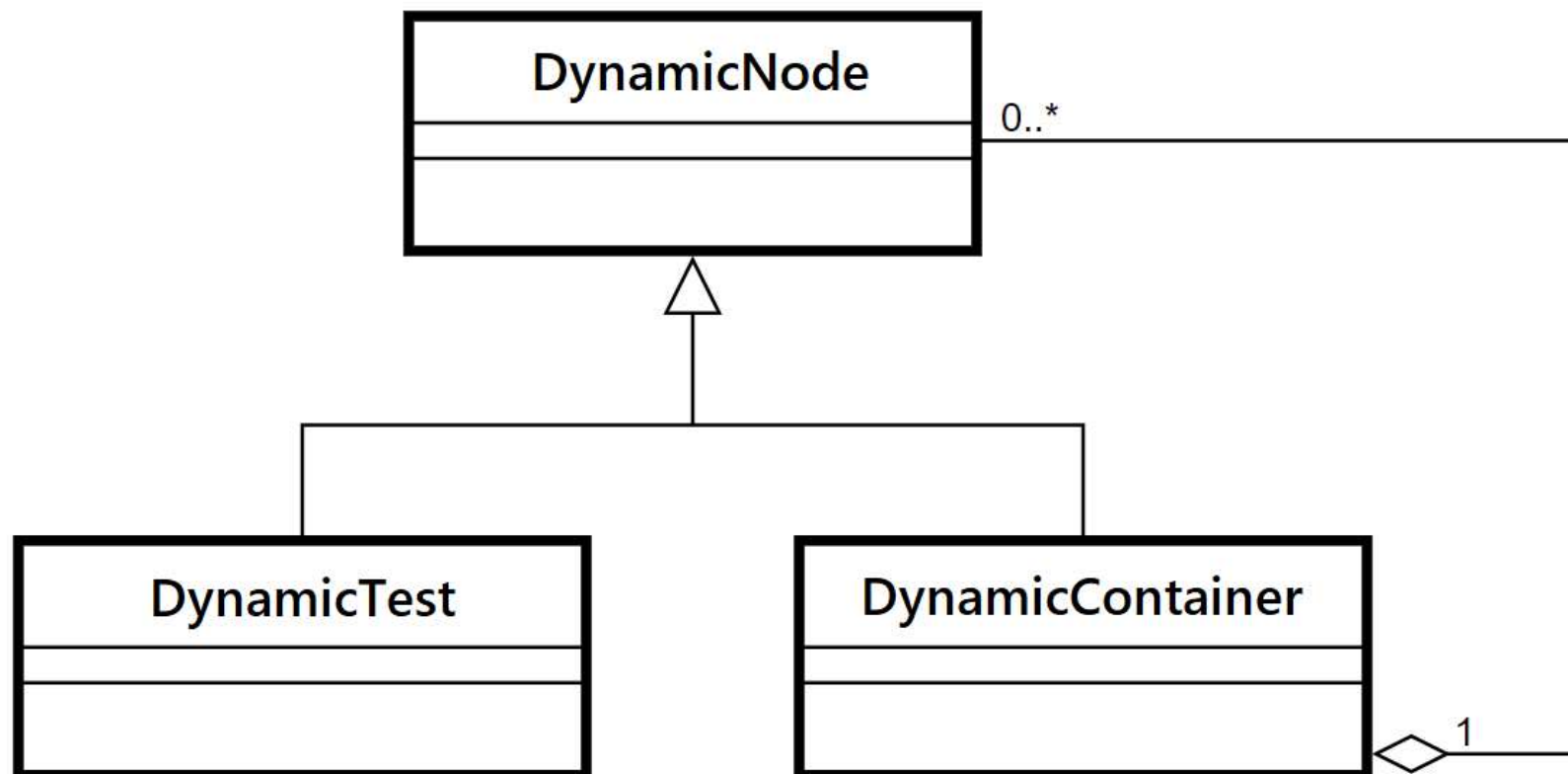
```
class CalculatorTest {  
    ...  
  
    @ParameterizedTest(name = "result of {0} x {1} should be {2}")  
    @MethodSource("createMulArgs")  
    public void mulTest(double a, double b, double expectedAnswer) {  
        assertEquals(expectedAnswer, calculator.mul(a,b));  
        assertEquals(expectedAnswer, calculator.ans);  
    }  
  
    private static Stream<Arguments> createMulArgs() {  
        return Stream.of(  
            Arguments.of(5, 0, 0),  
            Arguments.of(7, 4, 28),  
            Arguments.of(3, 2, 6)  
        );  
    }  
    ...  
}
```

| | |
|-----------------------------------|-------|
| ✓ Test Results | 34 ms |
| ✓ CalculatorTest | 34 ms |
| ✓ mulTest(double, double, double) | 34 ms |
| ✓ result of 5 x 0 should be 0 | 31 ms |
| ✓ result of 7 x 4 should be 28 | 2 ms |
| ✓ result of 3 x 2 should be 6 | 1 ms |

Dynamische Tests mit @TestFactory:

- Eigenen sich insbesondere für hierarchische bzw. baumartige Datenstrukturen
- Aufbau der Tests entspricht dem Aufbau der Datenstruktur
- Klasse DynamicTest, für das Testen des Verhaltens eines Blatt-Elements
- Klasse DynamicContainer, für das Testen des Verhaltens eines Unterbaums
- Klasse DynamicNode, ein gemeinsames Interface zwischen DynamicTest und DynamicContainer

Dynamische Tests mit @TestFactory:



Dynamische Tests mit @TestFactory:

```
public class Addition implements ArithmeticExpression{

    ArithmeticExpression exp1;
    ArithmeticExpression exp2;

    public ArithmeticExpression getExp1() {
        return exp1;
    }

    public ArithmeticExpression getExp2() {
        return exp2;
    }

    public Addition(ArithmeticExpression exp1, ArithmeticExpression exp2) {
        this.exp1 = exp1;
        this.exp2 = exp2;
    }

    @Override
    public int evaluate() {
        return exp1.evaluate() + exp2.evaluate();
    }

}
```

```
public class Number implements ArithmeticExpression{

    private final int value;

    public Number(int value){
        this.value = value;
    }

    @Override
    public int evaluate() {
        return this.value;
    }

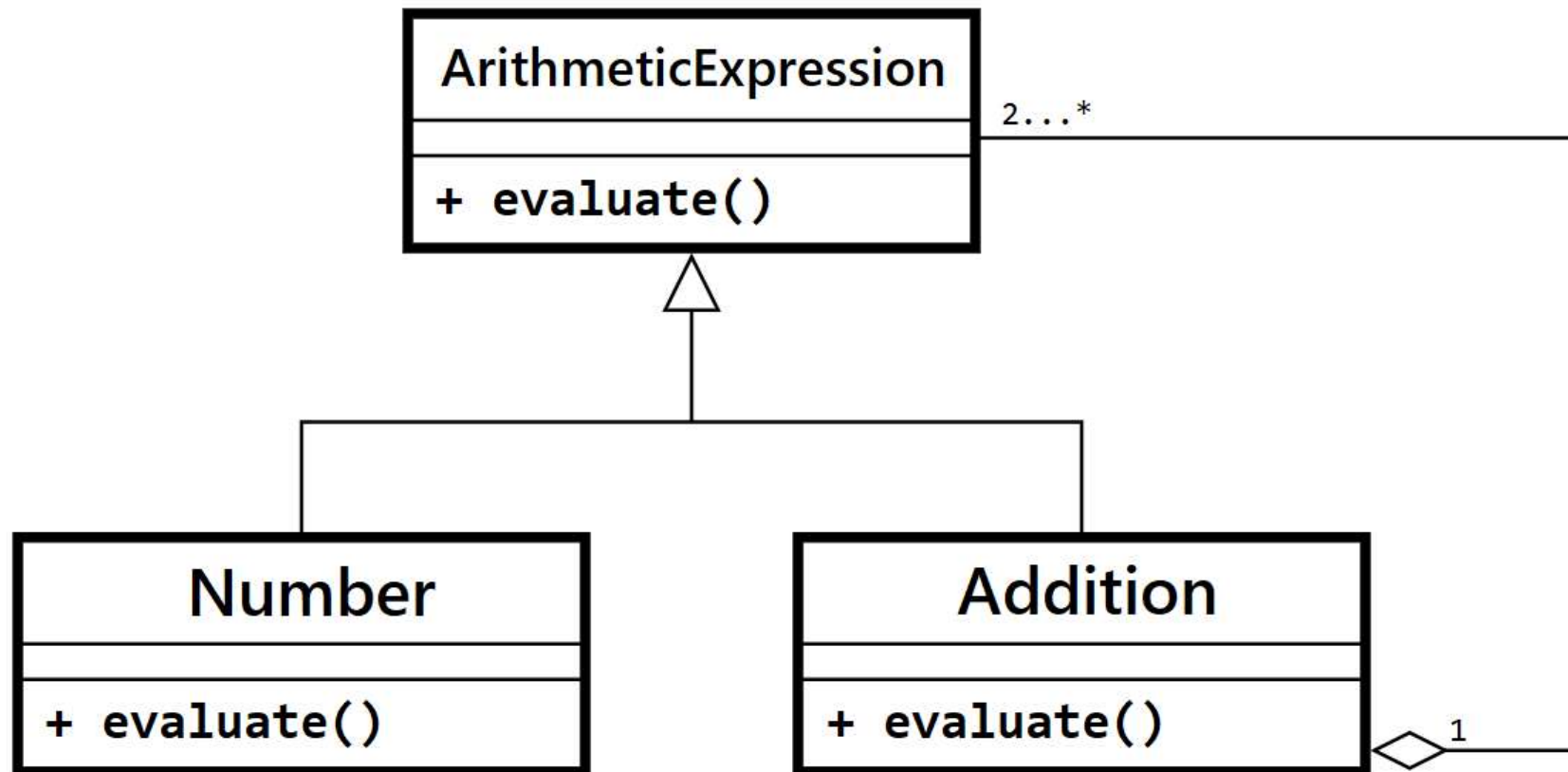
}
```

```
public interface ArithmeticExpression {

    int evaluate();

}
```

Dynamische Tests mit @TestFactory:



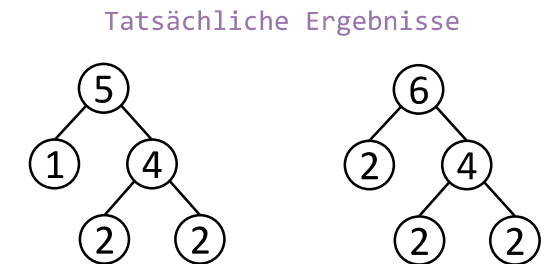
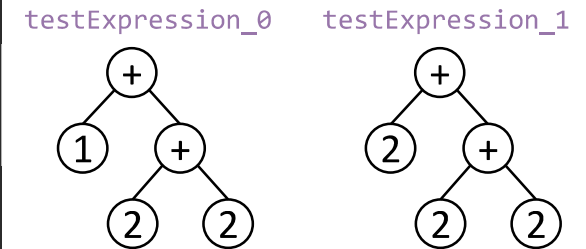
Dynamische Tests mit @TestFactory:

```
class ArithmeticExpressionTest {  
    // (1 + (2 + 2))  
    ArithmeticExpression testExpression_0 = new Addition(new Number(1), new Addition(new Number(2), new Number(2)));  
    // (2 + (2 + 2))  
    ArithmeticExpression testExpression_1 = new Addition(new Number(2), new Addition(new Number(2), new Number(2)));  
  
    List<Integer> expecteds_0 = List.of(5, 1, 4, 2, 2);  
    List<Integer> expecteds_1 = List.of(5, 1, 4, 2, 2);  
  
    int currentIndex = 0;  
    @TestFactory  
    public Stream<DynamicNode> test(){  
        DynamicNode dn_0 = buildTestTree(testExpression_0, expecteds_0);  
        // Position zuruecksetzen  
        currentIndex = 0;  
        DynamicNode dn_1 = buildTestTree(testExpression_1, expecteds_1);  
        return Stream.of(dn_0, dn_1);  
    }  
  
    public DynamicNode buildTestTree(ArithmeticExpression ae, List<Integer> expecteds) {  
        int expected = expecteds.get(currentIndex++);  
        if (ae.getClass() == Addition.class) {  
            return DynamicContainer.dynamicContainer("addition",  
                Stream.of(DynamicTest.dynamicTest("result should be " + expected, () -> {  
                    int actual = ae.evaluate();  
                    assertEquals(expected, actual);  
                })),  
                buildTestTree(((Addition) ae).getExp1(), expecteds),  
                buildTestTree(((Addition) ae).getExp2(), expecteds));  
        }  
        return DynamicTest.dynamicTest("number should be " + expected, () -> {  
            int actual = ae.evaluate();  
            assertEquals(expected, actual);  
        });  
    }  
}
```

Erwartete Ergebnisse

Führt 2 Testbäume aus

Erstellt die Testbäume



| | |
|--------------------------|-------|
| Test Results | 13 ms |
| ArithmeticExpressionTest | 13 ms |
| test() | 13 ms |
| addition | 11 ms |
| result should be 5 | 10 ms |
| number should be 1 | |
| addition | 1 ms |
| result should be 4 | |
| number should be 2 | 1 ms |
| number should be 2 | |
| addition | 2 ms |
| result should be 5 | 2 ms |
| number should be 1 | |
| addition | |
| result should be 4 | |
| number should be 2 | |
| number should be 2 | |

Code + IntelliJ Projekt:

- Finden Sie auf Github unter: <https://github.com/TonyArar/seminar.git>
- Weitere Details zu JUnit 5 finden Sie in der Seminararbeit