codecentric

# Software Testing mit JUnit 5

## Architektur und neue Features

# Reinhard Prechtl

codecentric AG
München

reinhard.prechtl@codecentric.de
@reinhard_codes

www.codecentric.de
www.reinhard.codes

# Disclaimer

- Ich bin nicht selbst im JUnit Projekt aktiv
- JUnit 5 ist *Work in Progress*
- https://github.com/junit-team/junit5/
- Mehr Infos u.a.:
  http://junit.org/junit5/docs/5.0.0-M2/user-guide/
  http://blog.codefx.org/tag/junit-5/
- Videos von Sam Brannen und Nicolai Parlog auf Youtube

# Warum JUnit 5?

- fast 20 Jahre alt
- Hervorragende Toolunterstützung
- aktuell Version 4.12
- Keine Unterstützung von modernen Sprachfeatures (Java 8)

- Erfolg und langsame Entwicklung bedingen sich gegenseitig

- One JAR to rule them all:
  - IDEs, Buildtools, Entwickler, Erweiterungen benutzen das gleiche Artefakt
  - Keine Tool-spezifische API, vieles ist mit Reflection umgesetzt

- Großer Erfolg von JUnit 4 erschwert Weiterentwicklung

## Erweiterungsmöglichkeiten

- Runners (seit 4.0):
  - Sehr mächtig, sehr schwergewichtig
  - keine Kombination möglich
  - bspw: `Parametrized, Categories`
- Rules (seit 4.7):
  - Eigene Objekte in Testablauf injizieren
  - Flexibler als Runner
  - MethodRules/ClassRules
  - Nur before/after
  - bspw: `ExpectedException, MockitoRule`

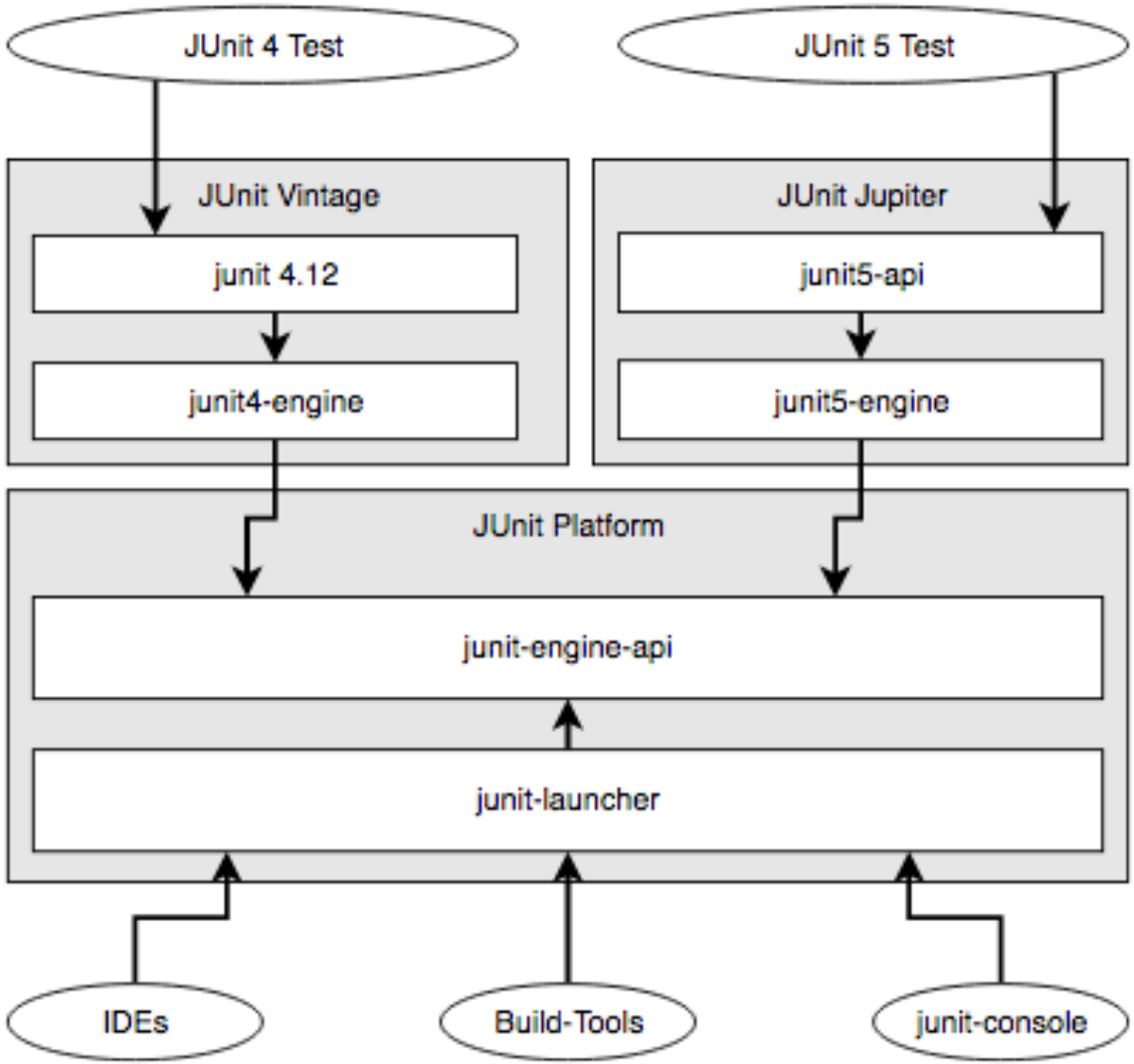
# JUnit 5

- JUnit 5 durch Crowdfunding gestartet
- Work in Progress
- Besonders im Fokus:
  - Modularisierung
  - Erweiterbarkeit
  - Verwendung neuer Sprachfeatures (JUnit Lambda)
  - Kompatibilität
- Erste Integration in IDEs und Buildtools

# JUnit 5 - Architektur

- Eine API um Tests zu schreiben
- Eine API um Tests zu finden und zu starten
- Verschiedene Engines um Tests auszuführen

➢ Seperation of Concerns



- JUnit5 Engine: Jupiter
  package: org.junit.jupiter
- JUnit4 Engine: Vintage
  package: org.junit.vintage
- ...andere Engines:
  - Spek (Kotlin)
  - Specsy (Scala, Groovy, Java)

- JUnit5 Tests durch JUnit4 ausführen:
  @Runner(JUnitPlatform.class)

- Modularer Aufbau
- Vorwärts- und rückwärtskompatibel
- Ermöglicht sanfte Migration
- Öffnet die gesamte Plattform

# JUnit 5 - Erweiterbarkeit

- *Extension Points* statt *Features*:
  - BeforeAll Callback
  - BeforeEach Callback
  - BeforeTest Execution
  - AfterTest Execution
  - AfterEach Callback
  - AfterAll Callback

  - Test Execution Condition / Container Execution Condition
  - TestInstance PostProcessor
  - Parameter Resolution
  - Exception Handling

## – Beispiel Zeitmessung

```
public class BenchmarkExtension implements
          BeforeTestExecutionCallback,
          AfterTestExecutionCallback {

 private long start;

 @Override
 public void beforeTestExecution(TestExtensionContext ctx) {
   start = currentTimeMillis();
 }
 @Override
 public void afterTestExecution(TestExtensionContext ctx) {
  System.out.println(
          "Test " + ctx.getDislayName() +
          " took " + currentTimeMillis() - start +" ms");
 }
}
```

- Beispiel konditionale Testausführung

```
public class DisabledWhenExternalSystemUnavailableExtension
 implements TestExecutionCondition {

 @Override
 public ConditionEvaluationResult
          evaluate(TestExtensionContext ctx) {

  if (externalSystemAvailable()) {
    return enabled("External system is available");
  } else {
    return disabled("External system isn't available!")
  }

 }
 // ...
}
```

- Anwendung einer Extension

```
@ExtendWith(BenchmarkExtension.class)
public class SomeTests {
    // ...
}
```

- ...oder mit Meta-Annotations

```
@ExtendWith(BenchmarkExtension.class)
public @interface Benchmarked {}

@Benchmarked
public class SomeTests {
    // ...
}
```

# Fazit:

- sehr flexibel
- gut kombinierbar
- Anpassbar durch Meta-Annotationen

# JUnit 5 – Neue Features

- Assertions
- Assumptions
- Nested tests
- Display names
- Dynamic tests
- Conditional tests

## - Nur Basis Assertions

```
assertTrue(service.isEnabled(), "Expected to be true")

assertEquals(expected, actual, () -> "That didn't work!")

assertAll("Adresstests",
  ( ) -> assertEquals("Nürnberg", addresse.getOrt()),
  ( ) -> assertEquals("90411", addresse.getPlz());
);

Exception ex = assertThrows(Exception.class,
        service::invoke);

assertTimeout(Duration.parse("PT0.500S"), service::invoke);
assertTimeoutPreemptivly(Duration.parse("PT0.500S"),
        service::invoke);
```

- Wer kennt Assumptions?

```
assumeTrue(service.isEnabled(), "Expected to be true");
assumeFalse(expected, actual, () -> "That didn't work!");
assumingThat(service.isEnabled(), () -> { /* Do something */ });
```

- Tests können verschachtelt werden

  @Nested

# JUnit 5 – Display Names

- Für Tests können Namen vergeben werden

```
@DisplayName("Eine Adresse sollte")
public class AddressTests {

    // ...

    @Test
    @DisplayName("einen gültigen Ort besitzen")
    public void shouldHaveValidCity() {
      assertEquals("Ort", addressService.getAddresse().getOrt());
    }

}
```

- Parametrisierte Tests in JUnit 4

```
@RunWith(Parameterized.class)
public class FibonacciTest {

    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {
                {0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }});
    }
    private int fInput;
    private int fExpected;

    public FibonacciTest(int input, int expected) {
        this.input= input;
        this.expected= expected;
    }

    @Test
    public void test() {
        assertEquals(expected, Fibonacci.compute(input));
    }
}
```

# JUnit 5 – Dynamic Tests

- Parametrisierte Tests mit JUnit 5 durch „Dynamic Tests“

```
public class AddressTests {

    @TestFactory
    public List<DynamicTest> createSomeTests() {
      return Arrays.asList(
            DynamicTest.dynamicTest(
              "This test is dynamically created",
              () -> assertEquals(1, 1, "One should be one")),
            DynamicTest.dynamicTest(
              "This test is also dynamically created",
              () -> assertEquals(2, 2, "One should be one")),
    }

}
```

# JUnit 5 – Conditional Tests

- Tests können deaktiviert werden

```
public class AddressTests {

    @Test
    @Disabled
    public void shouldHaveValidCity() {
      assertEquals("Ort", addressService.getAddresse().getOrt());
    }

}
```

- **ExecutionConditions** können deaktiviert werden!

- **Was fehlt noch?**
  - Lifecycle-Callbacks für dynamische Tests
  - Statische, parametrisierte Tests
  - Szenario Tests
  - Parallele Testausführung

## Zum Schluss...

- Neue Architektur öffnet die Plattform
- Viele neue Möglichkeiten durch Extensionpoints
- Tool-Support schon jetzt vorhanden
- JUnit 5 ist Gesprächsthema und wird sich durchsetzen

# Vielen Dank

@reinhard_codes