

# JUnit



State of the Union

# Görge Albrecht

Software Developer since 1989

Code Mentor - taking care of code

Need help writing simpler code? Contact me!

<http://code-mentor.com>

 @g\_o\_rge

# The Beginnings



# SUnit ⇒ JUnit

- SUnit written by Kent Beck around 1992
- JUnit created by Kent Beck and Erich Gamma around 1997
- JUnit became de facto standard of testing Java code

---

*Never in the field of software development  
was so much owed by so many to so few  
lines of code.*

---

– Martin Fowler

# JUnit 4

- steady progress up to JUnit 3.8.x
- big architectural change from framework style (subclassing) to DSL Style (annotations)
- **JUnit 4.4** added dependency to **Hamcrest**
- **JUnit 4.7** added Rules to setting the context of a test

# JUnit λ

- 10 Years since release of Junit 4
- Java 8 introduced Lambdas
  - not supported nor utilized by JUnit
- JUnit has always been a pet project
- ⇒ [Crowdfunding Campaign on Indiegogo](#)



# JUnit 5

- development started in Autumn 2015
- alpha release 2016-02-01
- three milestones in 2016
- M4 planned for 2017-04-01
- **JUnit 5.0 GA expected Q3 2017**

# Vision

- Decouple test execution and reporting from test definition and provisioning
- Rethinking JUnit's extensibility story
- Making use of Java 8 features

# Lessons learned from JUnit 4

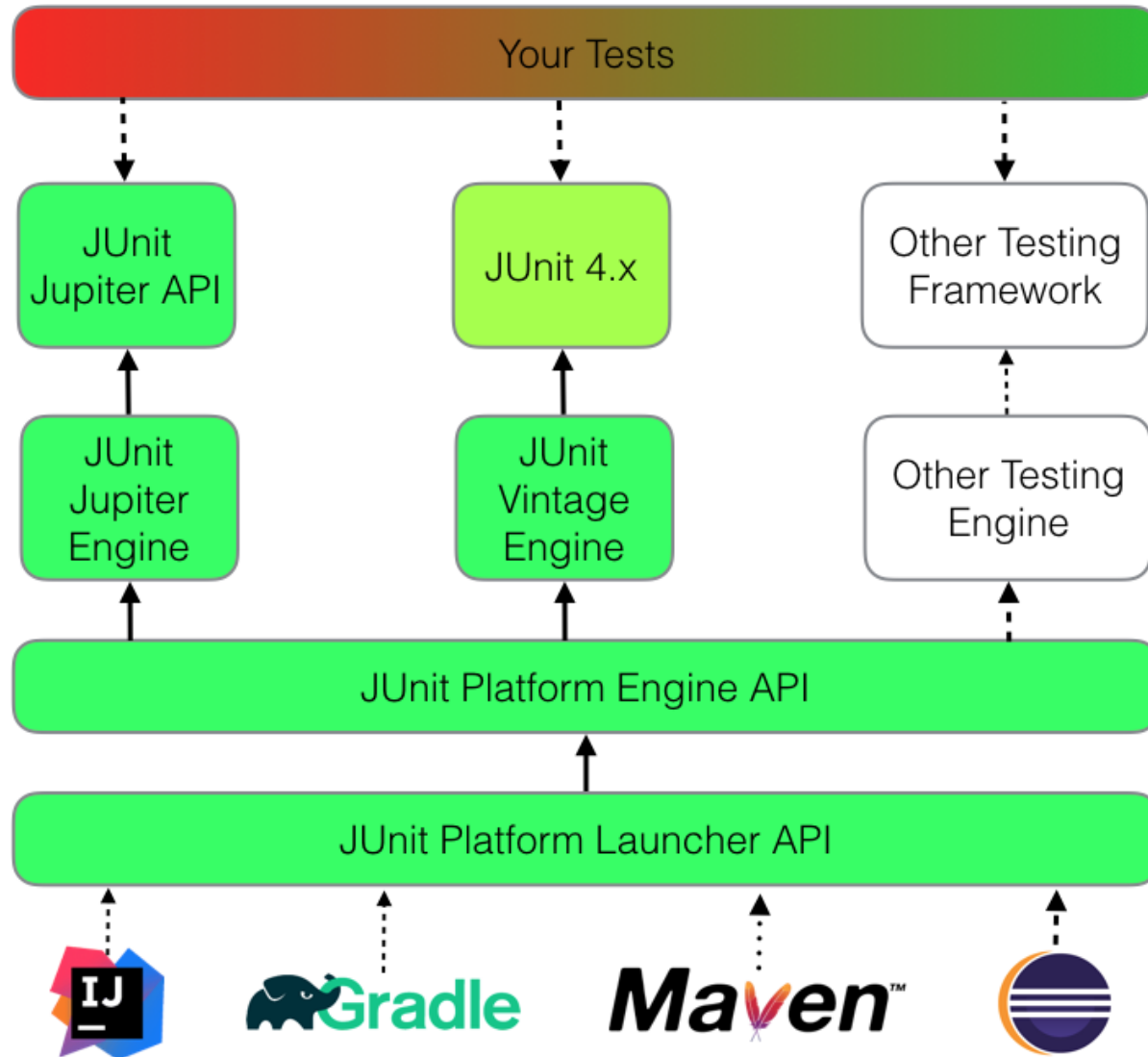
"If you release it, you're going to maintain it."

- Parameterized
- Hamcrest

# JUnit 5 Core Principles

- Prefer extension points over features
  - an extension point should be good at one thing
- It should be hard to write tests that behave differently based on how they are run
- Tests should be easy to understand
- Minimize dependencies (especially third-party)

# Components



# Syntactic sugar

# no public anymore

```
package com.codementor.junit5;

import org.junit.jupiter.api.Test;

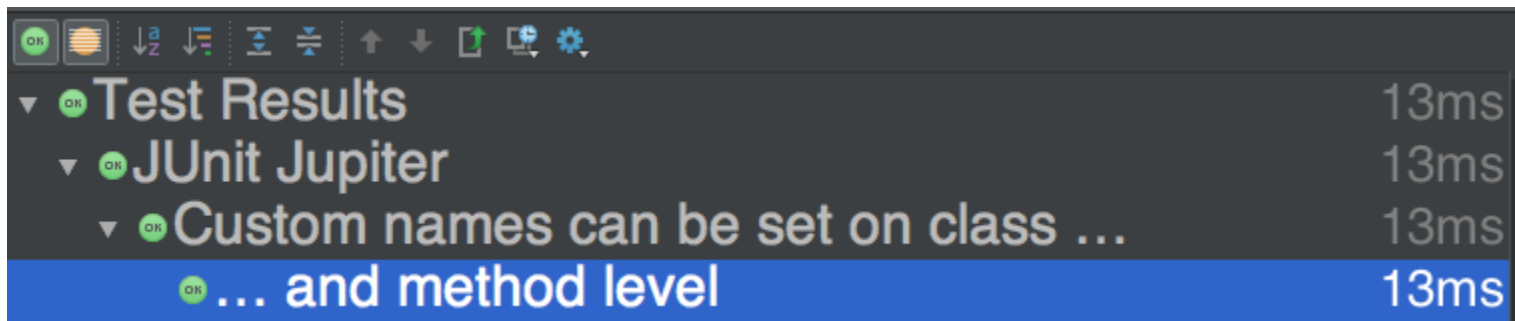
class NoPublicTest {

    @Test
    void name() {
    }
}
```

# Display Names

```
@DisplayName("Custom names can be set on class ...")
class DisplayNamesTest {

    @Test
    @DisplayName("... and method level")
    void name() {
    }
}
```



OK	Test Results	13ms
OK	JUnit Jupiter	13ms
OK	Custom names can be set on class ...	13ms
OK	... and method level	13ms



# Contract Tests I

```
class ContractTest implements EqualsTester<String> {  
  
    @Override  
    public String value() {  
        return "one";  
    }  
  
    @Override  
    public String equalValue() {  
        return "one";  
    }  
  
    @Override  
    public String nonEqualValue() {  
        return "two";  
    }  
}
```

# Contract Tests II

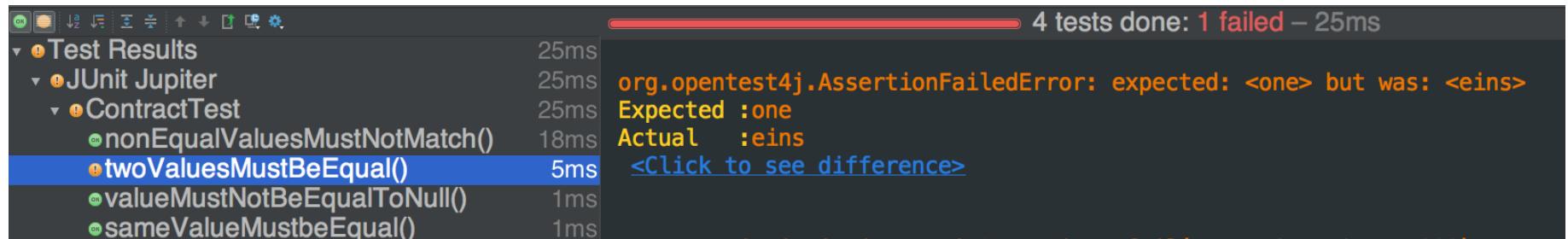
```
interface EqualsTester<T> {  
  
    T value();  
  
    T equalValue();  
  
    T nonEqualValue();  
  
    @Test  
    default void twoValuesMustBeEqual() {  
        assertEquals(value(), equalValue());  
    }  
  
    @Test  
    default void nonEqualValuesMustNotMatch() {  
        assertNotEquals(value(), nonEqualValue());  
    }  
}
```

# Contract Tests III

```
@Test
default void valueMustNotBeEqualToNull() {
    assertNotEquals(value(), null);
}

@Test
default void sameValueMustbeEqual() {
    assertEquals(value(), value());
}
}
```

# Contract Tests IV



The screenshot shows an IDE's test runner interface. At the top right, a progress bar indicates '4 tests done: 1 failed - 25ms'. The left sidebar shows a tree view of test results:

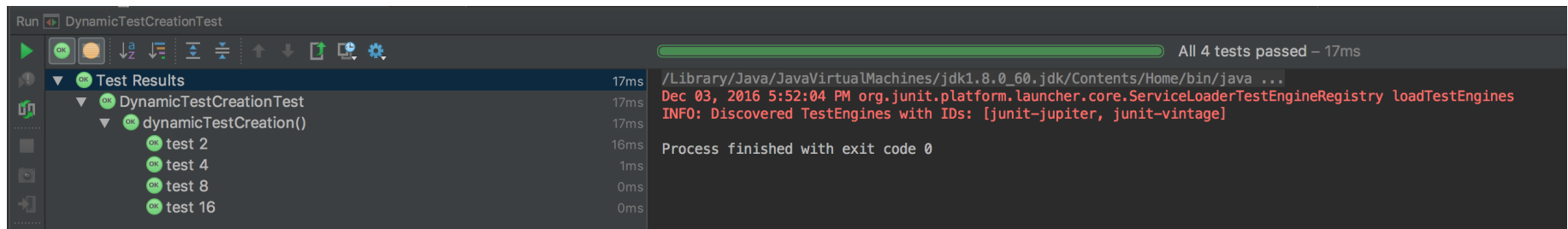
- Test Results (25ms)
  - JUnit Jupiter (25ms)
    - ContractTest (25ms)
      - nonEqualValuesMustNotMatch() (18ms)
      - twoValuesMustBeEqual() (5ms)**
      - valueMustNotBeEqualToNull() (1ms)
      - sameValueMustbeEqual() (1ms)

The right pane displays the error details for the failed test:

```
org.opentest4j.AssertionFailedError: expected: <one> but was: <eins>  
Expected :one  
Actual   :eins  
<Click to see difference>
```

# Dynamic Test Creation

```
@TestFactory
Stream<DynamicTest> dynamicTestCreation() {
    return IntStream.iterate(2, n -> n * 2).limit(4).mapToObj(
        n -> dynamicTest("test " + n, () -> assertTrue(n % 2 == 0))
    );
}
```



The screenshot shows the Run window of an IDE. At the top, a progress bar indicates "All 4 tests passed - 17ms". Below this, the "Test Results" section is expanded to show the following structure:

- DynamicTestCreationTest (17ms)
  - dynamicTestCreation() (17ms)
    - test 2 (16ms)
    - test 4 (1ms)
    - test 8 (0ms)
    - test 16 (0ms)

The right side of the window displays the following output:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
Dec 03, 2016 5:52:04 PM org.junit.platform.launcher.core.ServiceLoaderTestEngineRegistry loadTestEngines
INFO: Discovered TestEngines with IDs: [junit-jupiter, junit-vintage]
Process finished with exit code 0
```

# Nested Test

```
@DisplayName("A new int...")
class NestedClassesTest {

    private int count = MIN_VALUE;

    @BeforeEach
    void setCountToZero() {
        count = 0;
    }

    @Test
    @DisplayName("... is almost nothing..")
    void countIsZero() {
        assertEquals(0, count);
    }

    @Nested
    @DisplayName(" but after a step down ")
```

# Nested Test Output

The screenshot shows the Run console of an IDE. The title bar reads "Run NestedClassesTest". Below the title bar is a toolbar with various icons: a green play button, a green circle with "OK", a sun icon, a sort icon with "a/z", a list icon, a zoom icon, a refresh icon, a search icon, a refresh icon, a refresh icon, a refresh icon, and a gear icon. The main area of the console displays a tree view of test results. The root node is "Test Results" with a green "OK" icon and a duration of "22ms". It is expanded to show "JUnit Jupiter" (22ms), which is further expanded to show "A new int..." (22ms). This node is expanded to show five sub-nodes: "... is almost nothing..." (15ms), "... but after a step down..." (7ms), "... it grows..." (4ms), "... and in the end..." (3ms), and "... it's getting really BIG!" (3ms). All sub-nodes have a green "OK" icon.

Test Name	Duration
Test Results	22ms
JUnit Jupiter	22ms
A new int...	22ms
... is almost nothing...	15ms
... but after a step down...	7ms
... it grows...	4ms
... and in the end...	3ms
... it's getting really BIG!	3ms

# Assertions



# Message comes last, finally!

```
@Test
void messageComesLast() {
    org.junit.Assert.assertEquals(
        "JUnit <=4 message comes first", "aString", "aString");

    org.junit.jupiter.api.Assertions.assertEquals(
        "aString", "aString", "JUnit 5 message comes last");
}
```

# Lambdas FTW

```
@Test
void assertWithBooleanClassic() {
    assertTrue(true);
}

@Test
void assertWithBooleanLambda() {
    assertTrue(() -> true);
}

@Test
void assertWithBooleanMethodReference() {
    assertTrue(TRUE::booleanValue);
}
```

# Exceptions

```
@Test
void exception() {
    assertThrows(RuntimeException.class, this::illegal);
}

private void illegal() {
    throw new IllegalArgumentException("illegal");
}
```

# Exception Message Assertion

```
@Test
void exceptionMessageAssertion() {
    final RuntimeException exception =
        assertThrows(RuntimeException.class, this::illegal);
    assertEquals("illegal", exception.getMessage());
}

private void illegal() {
    throw new IllegalArgumentException("illegal");
}
```

# Lazy message evaluation

```
@Test
void assertWithLambdaMessage() {
    assertTrue(
        true, () -> "JUnit 5" + " message are evaluated " + "lazily");
}
```

# Assert all properties at once

```
@Test
void assertAllProperties() {
    Address address = new Address("New City", "Some Street", "No");

    assertAll(
        () -> assertEquals("New City", address.city),
        () -> assertEquals("Irgendeinestraße", address.street),
        () -> assertEquals("Nr", address.number)
    );
}
```

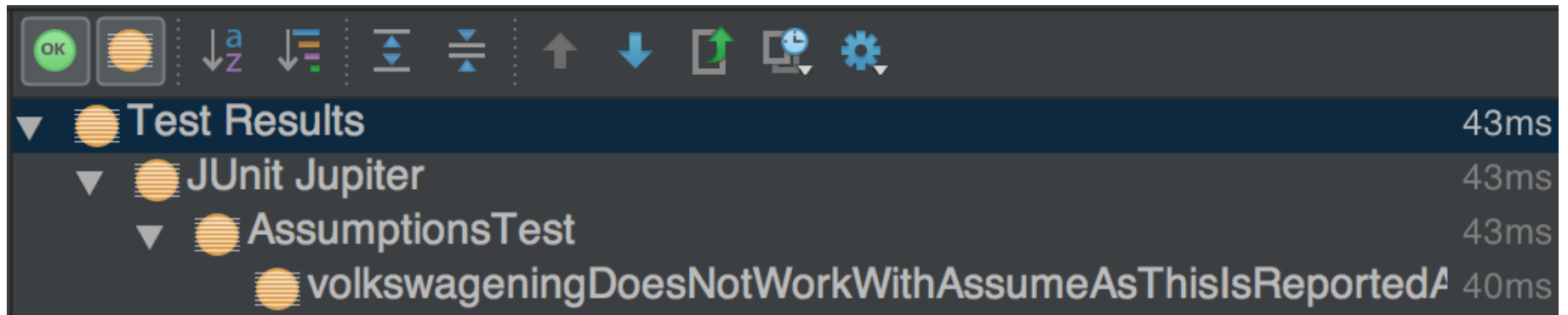
1 test failed – 28ms

Test Name	Duration	Failure Message
Test Results	28ms	
JUnit Jupiter	28ms	
AssertAllTest	28ms	
assertAllProperties()	28ms	org.opentest4j.MultipleFailuresError: Multiple Failures (2 failures) expected: <Irgendeinestraße> but was: <Some Street> expected: <Nr> but was: <No> <a href="#">&lt;Click to see difference&gt;</a>

# Assumptions

# Volkswagening

```
@Test
void volkswageningDoesNotWorkWithAssumeAsThisIsReportedAsAborted() {
    assertTrue(notOnTestStation());
    assertTrue(abgasWerteUnterVorgabe());
}
```



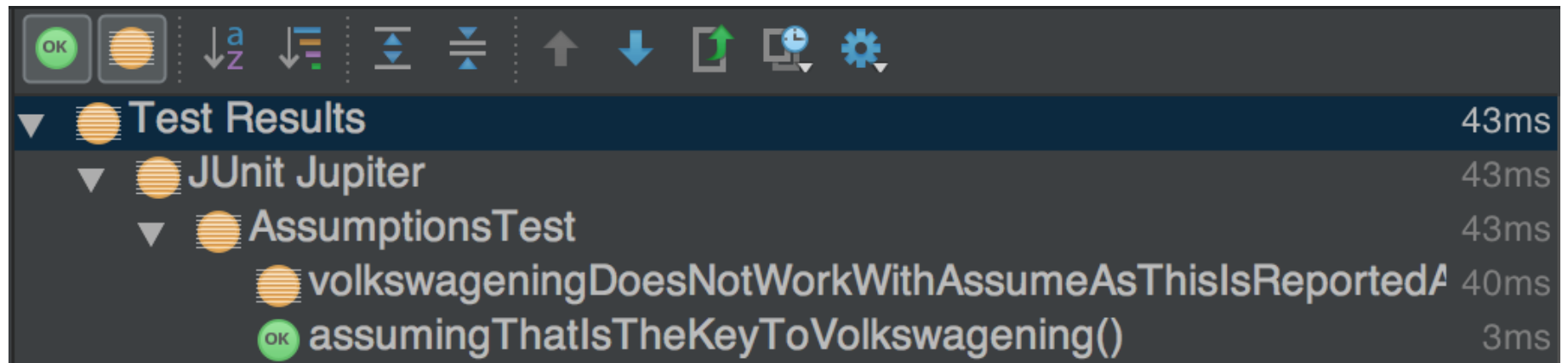
The screenshot shows the test results panel of an IDE. At the top, there is a toolbar with icons for 'OK', a failed test (orange circle with horizontal lines), sorting (a-z, list), zooming (vertical and horizontal arrows), and other actions (up/down arrows, refresh, search, settings). Below the toolbar, the test results are listed in a tree view:

Test Name	Duration
Test Results	43ms
JUnit Jupiter	43ms
AssumptionsTest	43ms
volkswageningDoesNotWorkWithAssumeAsThisIsReportedAsAborted	40ms



# real Volkswagening

```
@Test
void assumingThatIsTheKeyToVolkswagening() {
    assumingThat(notOnTestStation(),
        () -> assertTrue(abgasWerteUnterVorgabe()));
}
```



The screenshot shows the test results panel of an IDE. At the top, there is a toolbar with icons for 'OK', a test runner icon, sorting options (alphabetical, by duration), and other navigation tools. Below the toolbar, the test results are displayed in a tree view. The root node is 'Test Results' with a duration of 43ms. It contains a sub-node 'JUnit Jupiter' (43ms), which in turn contains 'AssumptionsTest' (43ms). Under 'AssumptionsTest', there are two test methods: 'volkswageningDoesNotWorkWithAssumeAsThisIsReportedA' (40ms) and 'assumingThatIsTheKeyToVolkswagening()' (3ms). The 'assumingThatIsTheKeyToVolkswagening()' method is marked as successful with a green 'OK' icon.

Test Method	Duration
Test Results	43ms
JUnit Jupiter	43ms
AssumptionsTest	43ms
volkswageningDoesNotWorkWithAssumeAsThisIsReportedA	40ms
assumingThatIsTheKeyToVolkswagening()	3ms

# Extensions & Dependency Injection

# Extensions

- no `Runner` and `@Rule` / `@ClassRule` anymore
- replaced by `Extensions`
- registered via `@ExtendWith` on class, method or annotation
- multiple extensions per element possible
- registered extensions are inherited

# Extensions example

```
@ExtendWith(MockitoExtension.class)
@ExtendWith(SpringExtension.class)
class MyTest {

    @Test void test1() {}

    @ExtendWith(TimingExtension.class)
    @Test void test2() {}
}
```

# Extension Types

- Conditional Test Execution
  - `ContainerExecutionCondition`
  - `TestExecutionCondition`
- Test Instance Post-processing
  - `TestInstancePostProcessor`  
(`MockitoExtension`, `SpringExtension`)
- Parameter Resolution
  - `ParameterResolver`

# Test Lifecycle Callbacks

- `BeforeAllCallback`
    - `BeforeEachCallback`
      - `BeforeTestExecutionCallback`
      - `AfterTestExecutionCallback`
    - `AfterEachCallback`
  - `AfterAllCallback`
- ⚡ `TestExecutionExceptionHandler`

# Dependency Injection

- JUnit 4.x: DI via Runner setting method parameters
- JUnit 4.7: DI via @Rule settings fields in test class
- JUnit 5: `ParameterResolver` FTW
  - multiple `ParameterResolver` per test method possible
  - each `ParameterResolver` can handle different parameter types

# DI example

```
class MyTest {  
  
    @ExtendWith(MockitoExtension.class)  
    @ExtendWith(VertxExtension.class)  
    @Test void test(@Mock MyService service, TestContext context) {}  
}
```



# Built-in Parameter Resolver

- `TestInfoParameterResolver`
  - provides test with context (name, tags, etc.) via `TestInfo` parameter
  - works also on lifecycle callbacks
- `TestReporterParameterResolver`
  - injects `TestReporter` into test
  - test can provide runtime information via `TestReporter` to execution env

# Migration from JUnit 4

# Remarks

- JUnit 5 is a complete rewrite
- neither compile nor binary compatibility
- bridges provided down- and upwards for smooth migration
- limited JUnit 4 rule support
- ⇒ no need to update all existing tests

# Execution

- run JUnit 5 based tests with JUnit 4 ?
  - JUnit 4 Runner: `JUnitPlatform`
- run JUnit 4 based tests with JUnit 5 ?
  - JUnit Platform engine: `junit-vintage-engine`

# Packages

JUnit < 4

`junit`

---

JUnit 4.x

`org.junit`

---

JUnit 5

`org.junit.jupiter.api`

# Assertions & Assumptions

## **JUnit 5**

---

`org.junit.jupiter.Assertions`

---

`org.junit.jupiter.Assumptions`

## **JUnit 4.x**

---

`org.junit.Assert`

---

`org.junit.Assume`

# Annotations

## JUnit 5

## JUnit 4.x

---

@Test (new package)

@Test

---

@TestFactory

-

---

@Disabled

@Ignore

---

@DisplayName

-

---

@Tag

@Category

# Lifecycle Annotations

## JUnit 5

@BeforeEach

@AfterEach

@BeforeAll

@AfterAll

@ExtendWith

## JUnit 4.x

@Before

@After

@BeforeClass

@AfterClass

~ @RunWith, @Rule,  
@ClassRule



Switch to JUnit 5 ?

# well ...

- no big improvements regarding Assertions API
  - use AssertJ or Hamcrest
- IDEs not there yet
  - [eclipse: beta branch](#)
  - IntelliJ: steady progress released
  - use maven/gradle in the meantime

# but ...

- new extension points will bring big opportunities 👍
- launcher and engine API
  - enables IDEs execute tests independent of the actual testing framework 👍

# Start today!

- analyze your code base and identify open ends
- convert your `Runner/Rule` to `Extension`
- ask your `Runner/Rule` provider to do so
- test your favorite IDE and file bugs
- give feedback to JUnit 5 team

# Thanks for listening

 @g\_o\_rge · <http://code-mentor.com>