# Junit

*14.-15.4.2016*

Ján Švantner
CIIT

# Junit 5

| | |
|---|---|
| @DisplayName | Displays custom name of the |
| @BeforeEach | @Before from JUnit4 |
| @AfterEach | @After from JUnit4 |
| @BeforeAll | @BeforeClass from JUnit4 |
| @AfterAll | @AfterClass from JUnit4 |
| @Nested | Nested tests |
| @Tags | Used for filtering the test by test runner |
| @Disabled | @Ignore from JUnit4 |
| @ExtendWith | Extension library: replacement for @Rule, @ClassRule, custom runners |
| | |

# Grouped assertions

- Message is now last parameter
- Grouped assertions

@Test

```
void groupedAssertions() {
        // In a grouped assertion all assertions are executed
        // failures will be reported together.
        assertAll("address",
            () -> assertEquals("John", address.getFirstName()),
            () -> assertEquals("User", address.getLastName())
        );
    }
```

# Tagging

```
@Tag("fast")
@Tag("model")
class TaggingDemo {

    @Test
    @Tag("taxes")
    void testingTaxCalculation() {
    }

}
```

# Nested tests

```
class TestingAStack {
    Stack<Object> stack;

    @Test
    @DisplayName("is instantiated with new Stack()")
    void isInstantiatedWithNew() {
        new Stack<Object>();
    }

    @Nested
    @DisplayName("when new")
    class WhenNew {

        @BeforeEach
        void init() {
            stack = new Stack<Object>();
        }

        @Test
        @DisplayName("throws EmptyStackException when popped")
        void throwsExceptionWhenPopped() {
            Assertions.expectThrows(EmptyStackException.class, () -> stack.pop());
        }
```

# Nested tests

```
@Nested
        @DisplayName("after pushing an element")
        class AfterPushing {

            String anElement = "an element";

            @BeforeEach
            void init() {
                stack.push(anElement);
            }

            @Test
            @DisplayName("it is no longer empty")
            void isEmpty() {
                Assertions.assertFalse(stack.isEmpty());
            }
        }
    }
}
```

# Method Parameters and DI

```
@ExtendWith(MockitoExtension.class)
class MyMockitoTest {

    @BeforeEach
    void init(@InjectMock Person person) {
        when(person.getName()).thenReturn("Dilbert");
    }

    @Test
    void simpleTestWithInjectedMock(@InjectMock Person person) {
        assertEquals("Dilbert", person.getName());
    }

}
```

# Interface Default Methods

```
public interface Testable<T> {
    T createValue();
}
public interface EqualsContract<T> extends Testable<T> {
    T createNotEqualValue();
    @Test
    default void valueDoesNotEqualNull() {
        T value = createValue();
        assertFalse(value.equals(null));
    }

    @Test
    default void valueDoesNotEqualDifferentValue() {
        T value = createValue();
        T differentValue = createNotEqualValue();
        assertNotEquals(value, differentValue);
        assertNotEquals(differentValue, value);
    }
}
```

# Running tests

- No IDE support
- Build systems:
  - Gradle
  - Maven
- Console Runner
- Using JUnit4

# Suites

```
@RunWith(JUnit5.class)
@Packages("example")
public class JUnit4SuiteDemo {
}
```

- Annotations:
  - @Classes
  - @ExcludeTags
  - @FilterClassName
  - @Packages
  - @RequireTags
  - @UniqueIds

# Extension Model

```
@ExtendWith({ FooExtension.class,
BarExtension.class })

class MyTestsV1 {

    // ...

}
```

- Replace the Rules, ClassRules
- Annotation processing
- Still experimental  !!!

# Extension points

- ContainerExecutionCondition
- TestExecutionCondition
- MethodParameterResolver
- Test Lifecycle Callbacks
  - BeforeEachExtensionPoint
  - AfterEachExtensionPoint
  - BeforeAllExtensionPoint
  - AfterAllExtensionPoint
- ExceptionHandlerExtensionPoint

# TDD

- Minify the 'test – develop' loop
- Testing first
- Leads to cleaner code
- Increases
- Developer is not stuck in endless analysis

# TDD

- Add a test
- Run all tests and see if the new one fails
- Write some code
- Run tests
- Refactor code
- Repeat

# The Bowling Game

- In this section, we will look at the example of a class that calculates the score for a game of bowling. The rules for this are as follows:
  - The game consists of 10 frames
  - In each frame the player has two opportunities to knock down 10 pins.
  - The score for a frame is the total number of pins knocked down, plus bonuses for strikes and spares.
  - A spare is when the player knocks down all 10 pins in two tries.
  - The bonus for that frame is the number of pins knocked down by the next roll.
  - A strike is when the player knocks down all 10 pins on the first try.
  - The bonus for that frame is the value of the next two balls rolled

# Hamcrest matchers

```
// JUnit 4 for equals check
assertEquals(expected, actual);
// Hamcrest for equals check
assertThat(actual, is(equalTo(expected)));

// JUnit 4 for not equals check
assertFalse(expected.equals(actual));
// Hamcrest for not equals check
assertThat(actual, is(not(equalTo(expected))));
```

# Matchers Chaining

```
assertThat("test", anyOf(is("testing"),
containsString("est")));
```

# Possibilities

- allOf - matches if all matchers match (short circuits)
- anyOf - matches if any matchers match (short circuits)
- not - matches if the wrapped matcher doesn't match and vice versa
- equalTo - test object equality using the equals method
- is - decorator for equalTo to improve readability
- hasToString - test Object.toString
- instanceOf, isCompatibleType - test type
- notNullValue, nullValue - test for null
- sameInstance - test object identity
- hasEntry, hasKey, hasValue - test a map contains an entry, key or value
- hasItem, hasItems - test a collection contains elements
- hasItemInArray - test an array contains an element
- closeTo - test floating point values are close to a given value
- greaterThan, greaterThanOrEqualTo, lessThan, lessThanOrEqualTo - test ordering
- equalToIgnoringCase - test string equality ignoring case
- equalToIgnoringWhiteSpace - test string equality ignoring differences in runs of whitespace
- containsString, endsWith, startsWith - test string matching

# List matchers

```
public class HamcrestListMatcherExamples {
    @Test
    public void listShouldInitiallyBeEmpty() {
        List<Integer> list = Arrays.asList(5, 2, 4);

        assertThat(list, hasSize(3));

        // ensure the order is correct
        assertThat(list, contains(5, 2, 4));

        assertThat(list, containsInAnyOrder(2, 4, 5));

        assertThat(list, everyItem(greaterThan(1)));

    }
}
```

# Custom matcher

```
public class RegexMatcher extends TypeSafeMatcher<String> {

  private final String regex;

  public RegexMatcher(final String regex) {
    this.regex = regex;
  }

  @Override
  public void describeTo(final Description description) {
    description.appendText("matches regular expression=`" + regex + "`");
  }

  @Override
  public boolean matchesSafely(final String string) {
    return string.matches(regex);
  }


   // matcher method you can call on this matcher class
    public static RegexMatcher matchesRegex(final String regex) {
        return new RegexMatcher(regex);
    }
}
```

# Usage custom matcher

```
package com.vogella.android.testing.applicationtest;


import org.junit.Test;

import static org.hamcrest.MatcherAssert.assertThat;

public class TestCustomMatcher {

    @Test
    public void testRegularExpressionMatcher() throws Exception {
        String s ="aaabbbaaaa";
        assertThat(s, RegexMatcher.matchesRegex("a*b*a*"));
    }

}
```

# Sources

- External sources:
- http://junit.org/junit5
- http://www.codeaffine.com/2016/02/18/junit-5-first-look/
- http://www.codeaffine.com/2016/04/06/replace-rules-in-junit5/

  http://www.vogella.com/tutorials/Hamcrest/article.html
- https://sites.google.com/site/tddproblems/all-problems-1
- Presentation sources:

  https://github.com/jansvantner/course-junit