Modultests mit JUnit
Softwarequalität in der komponentenbasierten Entwicklung

Simon Döring Michael Edelmann

26. April 2023

Agenda

1. Einleitung
2. JUnit 5 in Eclipse
3. Live-Demo
4. Funktionen von JUnit
5. Fazit

Wiederholung: Unit-Tests vs. Integrationstests

- Unit-Tests
  - Isoliertes Testen einzelner Klassen und Methoden
  - Kann Fehler einfach und präsize aufdecken
  - Beispiel: „Funktioniert die Komponente für sich?“
- Integrationstests
  - Testen des Zusammenspiels mehrerer Softwarebausteine
  - Kann das Big-Picture der Applikation besser validieren
  - Beispiel: „Funktioniert das Zusammenspiel der Komponenten?“

Äquivalenzklassentests

- Annahme: Zu testende Funktionalität verhält sich für bestimmte Klasse von Werten gleich
- Tests müssen dann nur für wenige Stichproben aus dieser Äquivalenzklasse ausgeführt werden
- Beispiel: Monat
  - Werte zwischen 0 bis 11 gültig
  - Alle anderen Werte ungültig
  - Beispielhaft zu testende Werte: -5, -1, 0, 5, 11, 12, 17

JUnit

- Testing-Framework für Java und JVM
- Release der fünften Version 2017
- Einfache und intuitive Verwendung
- Ausgereift und viele fortgeschrittene Features

Hinweise für die Verwendung von JUnit 5 in Eclipse

- In den Properties des Projekts muss JUnit 5 dem **Modulepath** hinzugefügt werden
  - Dafür in den Projekt-Properties nach „Java Build Path“ navigieren, dann oben auf „Libraries“ klicken, in der Liste auf „Modulepath“ klicken und JUnit rechts über „Add Library“ hinzufügen
  - Eclipse fügt JUnit standardmäßig dem **Classpath** hinzu
- In module-info.java zwischen den geschweiften Klammern die Zeile `requires org.junit.jupiter.api;` hinzufügen
- Ausführen von Tests: Run → Run As → JUnit Test

Live-Demo

BooleanSupplier und Supplier<String>

- Functional Interfaces die jeweils `boolean` und `String` liefern
  - Kurzschreibweise durch Lambda-Ausdruck
- `BooleanSupplier` kann in JUnit-Assertions statt `boolean` verwendet werden
- `Supplier<String>` kann in JUnit-Assertions für Fehlermeldungen verwendet werden
  - Verwendung wenn Fehlernachricht nur im Fehlerfall erstellt werden kann

Anwendungsbeispiel Supplier<String>

```
import java.util.HashMap; import java.util.function.Supplier;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions.*;

class HashMapTest {
  @Test
  void testLeeresGet() {
    HashMap<String, String> map = new HashMap<>();
    String str = map.get("nicht-existenter schlüssel");
    Supplier<String> fehlerSupplier =
      () -> String.format("'%s' gefunden", str);
    assertNull(str, fehlerSupplier);
  }
}
```

Exception Testing mit assertThrows

- Testen ob bestimmte Benutzung korrekt Exceptions wirft

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions.*;

class ArrayTest {
  @Test
  void testOutOfBoundsIndexing() {
    int[] arr = new int[3];
    assertThrows(IndexOutOfBoundsException.class,
      () -> { int num = arr[5]; });
  }
}
```

DisplayName

- Spezielle Bezeichnung für Test-Klassen und -Methoden
- Ohne Namenskonvention
- Anzeige im Testbericht oder in IDEs

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

@DisplayName("Ein spezieller Testfall")
class DisplayNameTest {

  @Test
  @DisplayName("Eigener Testname, der Leerzeichen enthält")
  void testMitDisplayNameDerLeerzeichenEnthählt() { }
}
```

Assertions

- Annahmen, die Erfolg oder Misserfolg bestimmen
- Ergebnis einer Methode, wird mit einem erwarteten Wert verglichen

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
@Test
void testAssertions() {
  assertEquals(2, 1+1);
  assertNotNull(new Object());
}
```

Timeout

- Durchführung innerhalb einer festen Zeitspanne

```
import static java.time.Duration.ofMinutes;
import static org.junit.jupiter.api.Assertions.assertTimeout;
@Test
void testTimout() {
  assertTimeout(ofMinutes(2), () -> {
    // Lambda-Ausdruck mit Ausgabe, die innerhalb
    // von 2 Minuten ausgeführt werden soll
  }
}
```

Assumption

- Entscheidet, ob ein Test überhaupt ausgeführt werden soll
- JUnit Jupiter (JUnit 5) enthält zusätzliche Assumption-Methoden

```
import static org.junit.jupiter.api.Assumptions.assumeTrue;
@Test
void testNurAufCIServer() {
  assumeTrue("CI".equals(System.getenv("ENV")));
  // Rest des Tests
}
@Test
void testNurAufEntwicklungsUmgebungen() {
  assumeTrue("DEV".equals(System.getenv("ENV")),
    () -> "Test abbrechen: keine Entwicklungs-Umgebung");
  // Rest des Tests
}
```

Dependency Injection

- Test-Methoden und -Konstruktoren können Parameter haben
- `ParameterResolver`: API für Test-Erweiterungen,
  mit dynamischer Parameterauflösung zur Laufzeit
- Vordefinierte Resolver, die direkt verwendet werden können
  - `TestInfoParamterResolver`
    - Parameter-Typ `TestInfo`
  - `RepetitionInfoParameterResolver`
    - Parameter-Typ `RepetitionInfo` in `@RepeatedTest`, `@BeforEach`, `@AfterEach`
    - Enthält Informationen über die aktuellen Wiederholung
      und die Gesamtzahl eines sich wiederholenden Test (Repeated Test)
  - `TestReporterParameterResolver`
    - Parameter-Typ `TestReporter`

Dependency Injection

TestInfo

- Instanz entspricht dem aktuellen Container oder Test
- Enthält Informationen wie den Display Namen, die Klasse, Methode und hinzugefügte Tags

```
import org.junit.jupiter.api.TestInfo;
import org.junit.jupiter.api.Tag;

@Test
@DisplayName("TEST 1")
@Tag("my-Tag")
void test1(TestInfo testInfo) {
  assertEquals("TEST 1", testInfo.getDisplayName());
  assertTrue(testInfo.getTags().contains("my-tag"));
}
```

Dependency Injection

TestReporter

- Zusätzliche Informationen für Test-Berichte über den aktuellen Durchlauf

```
import org.junit.jupiter.api.TestReporter;

@Test
void reportValue(TestReporter testReporter) {
  testReporter.publishEntry("eine Statusmeldung");
  testReporter.publishEntry("ein Schlüssel", "ein Wert");
  Map<String, String> werte = new HashMap<>();
  values.put("Benutzername", "benutzername");
  values.put("Geburtsjahr", "1974");
  testReporter.publishEntry(werte);
}
```

Repeated Tests

- Test wird komplett eine definierte Anzahl wiederholt
- Kompletter Test-Lebenszyklus (`@BeforeEach`, `@Test`, `@AfterEach`)
- Wie wiederholte Definition von `@Test`

```
@RepeatedTest(10)
void wiederholenderTest() {
  // führt wiederholenderTest 10 Mal aus
}
```

Parametrisierte Tests

- Test mehrfach mit verschiedenen Werten ausführen
- Notation mit `@ParameterizedTest` anstatt `@Test`
- Erfordert mindestens eine *source* mit Werten für jeden Durchlauf

```
@ParameterizedTest
@ValueSource(strings = {"bob", "radar"})
void palindromes(String kandidat) {
  assertTrue(StringUtils.isPalindrome(kandidat));
}
```

Test-Templates

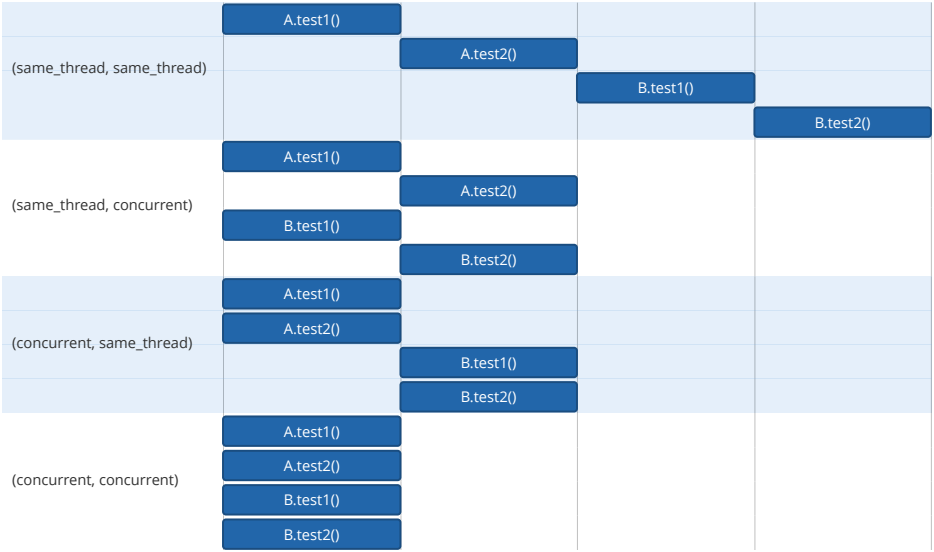
- Kein regulärer Testfall, eher eine Vorlage für Testfälle
- Mehrfacher Aufruf, durch einen registrierten Anbieter, wie ein normaler `@Test`
- Verbindung mit der `TestTemplateInvocationContextProvider`
- Repeated Test und Parametrisierte Tests sind spezielle Test-Templates

Paralleles Testen

- Experimentelle Funktion seit JUnit 5.3
- `junit.jupiter.execution.parallel.enabled = true`
- `junit.jupiter.execution.parallel.mode = concurrent`
- `junit.jupiter.execution.parallel.classes.default = concurrent`
- Concurrent-Mode auch über Annotation einstellbar
- Thread-Sicherheit über `@ResourceLock`

```
@Execution(CONCURRENT)
class GeteilteRessourceTest {
  @Test
  @ResourceLock(value = SYSTEM_PROPERTIES, mode = READ)
  void getGeteilteRessource() {
    assertNull(System.getProperty("null"));
  }
```

Parallele Test-Modi



Fazit

- Unit-Tests testen Klassen und Methoden
- JUnit ermöglicht einfache Unit-Tests in Java
- Eclipse unterstützt nativ JUnit 5
- Durch fortgeschrittene Funktionalitäten skaliert JUnit auch für größere Projekte
- Unit-Tests garantieren nicht die Funktionalität der Applikation im Ganzen

Quellen

- https://junit.org/junit5/
- https://junit.org/junit5/docs/current/user-guide
- https://junit.org/junit5/docs/current/user-guide/images/writing-tests_execution_mode.svg
- http://kleuker.iui.hs-osnabrueck.de/CSI/Methoden/kombiquAequivalenzklassenanalyse.html
- https://www.eclipse.org/community/eclipse_newsletter/2017/october/article5.php
- https://www.agilealliance.org/glossary/tdd/

Folien, Quellcode Live-Demo

https://github.com/sdoering01/junit-presentation

Vielen Dank für Ihre Aufmerksamkeit!
Haben Sie noch Fragen?