ToiletPaper #107

Besseres Testen, besseres Leben – mit AssertJ

Autor: Alexandros Panagiotidis / Senior Software Architect / Office Stuttgart

✗ Problem

Schnell – wie viele Varianten gibt es, um in JUnit zu prüfen, ob eine Collection leer bzw. nicht-leer ist? Sicherlich ist euch sofort `assertTrue(foo.isEmpty())` eingefallen. Aber ist euch schon mal `assertEquals(0, foo.size())` oder `assertFalse(foo.size() > 0)` vor die Reviewerflinte gelaufen? Überlegt euch jetzt mal, auf wieviele Arten man den Inhalt einer Collection mit einer anderen vergleichen kann. Richtig witzig wird es, wenn die Reihenfolge der Elemente eine Rolle spielt (oder nicht!).

Wem jetzt noch nicht die Haare zu Berge stehen, der möge überlegen, wie man Objekte inhaltlich vergleicht, wenn `equals()` nicht oder falsch implementiert wurde. All dies lässt sich einfach und elegant mit einem ausdrucksstarken Framework umsetzen – z. B. AssertJ.

➔ Lösung und Beispiele

Dank Fluent-Interfaces könnt ihr mit AssertJ eure Erwartungen einfach aneinanderreihen, um flott und übersichtlich mehrere Eigenschaften zu prüfen.

```
assertThat(jambit.getMission()).isNotNull().isEqualTo("100% Begeisterung");
```

Durch Assertions auf typisierte Objekte werden einige Fehlerklassen beim Erstellen der Tests direkt erschlagen. Bei Mengen lassen sich z.B. Größe und Inhalte einfach (und semantisch eindeutig) prüfen:

```
assertThat(Arrays.asList("Where", "innovation", "works")).hasSize(3)
    .containsExactly("Where", "innovation", "works") // Reihenfolge wichtig
    .containsOnlyOnce("works", "Where", "innovation"); // Reihenfolge egal
```

Ob Prädikate für eine Collection erfüllt sind, lässt sich elegant prüfen:

```
assertThat(jambit).extracting(Jambit::getEmployees)
    .allMatch(JambitEmployee::isTopOfMind); // alternativ {any,none}Match
```

Mittels `extracting()` lassen sich einzelne Felder und Ergebnisse von Methoden einfach verifizieren:

```
assertThat(jambit)
    .extracting(Jambit::getBosses).extracting("nickname", JambitBoss::getFullName)
    .containsOnly(tuple("Felli", "Peter F. Fellinger"), tuple("Harti", "Markus
Hartinger"));
```

Wie man sieht, lassen sich auch Namen der Felder statt Accessoren verwenden. Neben einzelnen Werten lassen sich auch Objekte miteinander vergleichen, bei denen `equals()` fehlt:

```
assertThat(jambitStuttgart).extracting(JambitLocation::getAddress)
    .isEqualToComparingFieldByFieldRecursively(JAMBIT_STUTTGART_EXPECTED_ADDRESS);
```

Für Exceptions gibt es auch einige nützliche Matcher:

```
assertThatThrownBy(() -> throw DeveloperException("Out of coffee"))
    .isInstanceOf(DeveloperException.class).hasMessage("Out of coffee");
```

Zum Abschluss ein komplexeres Beispiel, bei dem eine Collection mit einer Zweiten paarweise verglichen wird und dabei für jedes Paar alle Assertions ausgewertet werden:

```
assertThat(jambit).extracting(getDivisions)
    .zipSatisfy(fetchDivisionNames(), (JambitDivision actualDivision, String
expectedDivisionName) -> {
        try (AutoCloseableSoftAssertions softly = new
AutoCloseableSoftAssertions()) {

softly.assertThat(actualDivision).extracting("name").startsWith(expectedDivisionNa
me);

softly.assertThat(calculatePowerLevel(actualDivision)).isGreaterThan(9000);
        }
    });
```

✚ Weiterführende Aspekte

Wie ihr seht, hält AssertJ einiges für euch bereit. Am besten schaut ihr direkt mal in die Feature Highlights und legt dann sofort los. In der Dokumentation gibt es auch einige Migrationsguides – ihr könnt AssertJ allerdings auch parallel zu anderen Frameworks einsetzen.