Elegant Assertions

Assertj

Jürg Weilenmann, 15.10.2021

The challenge

- write a test checking some fields of items in list
- default assert lib is java assertions or Hamcrest

```
@Test
public void test2() {
    List<Watch> watches = getWatches();

    assertThat(watches, hasSize(3));

    Watch omega = new Watch("Omega", "Seamaster", 1000);
    Watch swatch = new Watch("Swatch", "Pink Lady", 99);
    Watch iwc = new Watch("IWC", "Aquatimer", 5000);
    assertThat(watches, containsInAnyOrder(omega, swatch, iwc));
}
```

- Failed, because Watch.equals() was not implemented
- It’s easy, just implement the equals(), isn’t it?

- BUT:
  - pull request was rejected by colleagues because equals is not used in production code
  - equals might be already implemented, but does not what you need to compare the fields (e.g. it’s an entity)
  - you don’t want to compare all fields, but only some of them

Options

- manually looping through the list an assert each field individually

```
Watch watch0 = watches.get(0);
assertThat(watch0.getBrand(), is(equalTo("Omega")));
assertThat(watch0.getModel(), is(equalTo("Seamaster")));
// ....
Watch watch1 = watches.get(1);
assertThat(watch1.getBrand(), is(equalTo("Swatch")));
assertThat(watch1.getModel(), is(equalTo("Pink Lady")));
```

- moving asserts into a method
- extend hamcrest with your own matcher

or give break to Assertj

Test with Assertj

```
@Test
public void test() {
    final List<Watch> watches = getWatches();
    assertThat(watches).asList()
            .hasSize(3)
            .extracting("brand", "model", "price")
            .contains(
                    tuple("Omega", "Seamaster", 10000),
                    tuple("Swatch", "Pink Lady", 99),
                    tuple("IWC", "Aquatimer", 5000)
            );
}
```

Look at this beauty ;-)

Dependency

```
<dependency>
  <groupId>org.assertj</groupId>
  <artifactId>assertj-core</artifactId>
  <version>3.21.0</version>
  <scope>test</scope>
</dependency>
```

- many options to compare

What I like on Assertj

- it has a fluent api

```
assertThat(frodo.getName())
        .startsWith("Fro")
        .endsWith("do")
        .isEqualToIgnoringCase("frodo");

  assertThat(fellowshipOfTheRing)
        .hasSize(9)
        .contains(frodo, sam)
        .doesNotContain(sauron);
```

-> great IDE support. Never have to remember the matchers name

- extracting fields (all or just the relevant ones)

```
   assertThat(watch)
        .extracting(Watch::getBrand, Watch::getModel, Watch::getPrice)
        .isEqualTo( Arrays.asList("Omega", „Seamaster“, 10000));
```

- field by field comparison

```
  assertThat(watch)
        .returns( "Omega", from(Watch::getBrand))
        .returns( "Seamaster", from(Watch::getModel));
```

What I like on Assertj

- compare by reflection and exclude fields

```
Watch watch1 = new Watch("Omega", "Speedmaster", 1000);
Watch watch2 = new Watch("Omega", "Speedmaster", 1000);
Watch watch3 = new Watch("Omega", "Speedmaster", 200);

assertThat(watch1)
      .usingRecursiveComparison()
      .isEqualTo(watch2)
      .isNotEqualTo(watch3)
      .ignoringFields("price")
      .isEqualTo(watch3);
```

you can ignore fields by name, type, regs and more

- builder for more complex, reusable comparators.

```
RecursiveComparisonConfiguration configuration = RecursiveComparisonConfiguration.builder()
                                                                                  .withIgnoredFields("hasPhd")
                                                                                  .build();

assertThat(doctors)
        .usingRecursiveFieldByFieldElementComparator(configuration)
        .contains(sheldon);
```

What I like on Assertj

- stream like api with filters for lists

```
assertThat(fellowshipOfTheRing)
  .filteredOn( character -> character.getName().contains("o") )
  .containsOnly(aragorn, frodo, legolas, boromir);
```

Many other ‚contains‘: doesNotContain, containsInAnyOrder, containsExactley, containsAll, ….

What I like on Assertj

- exceptions

```
// exception assertion, standard style ...
assertThatThrownBy(this::somethingIsGoingWrong)
  .hasMessage("boom!");

// ... or BDD style
Throwable thrown = catchThrowable(() -> this.somethingIsGoingWrong());
assertThat(thrown).hasMessageContaining(„boom");

// or
assertThatExceptionOfType(RuntimeException.class)
       .isThrownBy(this::somethingIsGoingWrong)
       .havingCause()
       .withMessage("boom!");
```

- and many more features
  - conditions (predicate)
  - BDD style (then() replaces assert())
  - assumption (conditional test execution: run test only on given condition)
  - dates, numbers, ranges

What I don’t like on Assertj

- extracting field values with lambdas not supported on all methods (some methods support it now)
- too many features (?)
- you can make things easily complex ;-)

References

- home of assertj: https://assertj.github.io/doc/
- most examples taken from https://assertj.github.io/doc/ (modified for readability)

Merçi for listening

Contact:

Mail: juerg.weilenmann@css.ch

Twitter: -

Facebook: -

LinkedIn: -

Instagram: -

WhatsApp: -

Git: -

BeachBar: 18:00 - 20:00