JUnit 5 Tutorial: Writing Assertions With JUnit 5 Assertion API

Petri Kainulainen March 17, 2018 6 comments

JUnit, JUnit 5

This blog post describes how we can write assertions by using the JUnit 5 assertion API. After we have finished this blog post, we:

- Can write basic assertions with JUnit 5.
- Know how we can customize the error message shown when an assertion fails.
- Understand how we can run multiple assertions as an assertion group.

Writing Assertions With JUnit 5

If we want to write assertions by using the “standard” JUnit 5 API, we must use the `org.junit.jupiter.api.Assertions` class. It provides `static` factory methods that we can use for writing assertions.

Before we will take a closer look at these methods, we have to know a few basic rules:

- If we want to specify a custom error message that is shown when our assertion fails, we have to pass this message as the last method parameter of the invoked assertion method.
- If we want to compare two values (or objects), we have to pass these values (or objects) to the invoked assertion method in this order: the expected value (or object) and the actual value (or object).

sertions with the `Assertions` class. Let's start by finding out
values.

true, we have to use the `assertTrue()` method of the
e to use this assertion:

yName;
;

```
5  import static org.junit.jupiter.api.Assertions.assertTrue;
6
7  @DisplayName("Write assertions for booleans")
8  class BooleanAssertionTest {
9
10     @Nested
11     @DisplayName("When boolean is true")
12     class WhenBooleanIsTrue {
13
14         @Test
15         @DisplayName("Should be true")
16         void shouldBeTrue() {
17             assertTrue(true);
18         }
19     }
20 }
```

If we want to verify that a `boolean` value is `false`, we have to use the `assertFalse()` method of the `Assertions` class. In order words, we have to use this assertion:

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Nested;
3  import org.junit.jupiter.api.Test;
4
5  import static org.junit.jupiter.api.Assertions.assertFalse;
6
7  @DisplayName("Write assertions for booleans")
8  class BooleanAssertionTest {
9
```

```
10      @Nested
```



```
alse")
```

```
se")
```

t an object is `null` or isn't `null`.

n't Null

If we want to verify that an object is `null`, we have to use the `assertNull()` method of the `Assertions` class. In other words, we have to use this assertion:

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Nested;
3  import org.junit.jupiter.api.Test;
4
5  import static org.junit.jupiter.api.Assertions.assertNull;
6
7  @DisplayName("Writing assertions for objects")
8  class ObjectAssertionTest {
9
10     @Nested
11     @DisplayName("When object is null")
12     class WhenObjectIsNull {
13
14         @Test
15         @DisplayName("Should be null")
16         void shouldBeNull() {
17             assertNull(null);
18         }
19     }
20 }
```

If we want to verify that an object isn't `null`, we have to use the `assertNotNull()` method of the `Assertions` class. In other words, we have to use this assertion:

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

                                          .Assertions.assertNotNull;

                                          r objects")


                                          t null")


                                           null")

                                          t());
19      }
20  }
```

Let's move on and find out how we can verify that two objects (or values) are equal or aren't equal.

Asserting That Two Objects or Values Are Equal

If we want to verify that the expected value (or object) is equal to the actual value (or object), we have to use the `assertEquals()` method of the `Assertions` class. For example, if we want to compare two `Integer` objects, we have to use this assertion:

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Nested;
3  import org.junit.jupiter.api.Test;
4
5  import static org.junit.jupiter.api.Assertions.assertEquals;
6
7  @DisplayName("Writing assertions for objects")
8  class ObjectAssertionTest {
9
10     @Nested
11     @DisplayName("When two objects are equal")
12     class WhenTwoObjectsAreEqual {
13
14         @Nested
15         @DisplayName("When objects are integers")
```

```
16          class WhenObjectsAreIntegers {
```



```
CTUAL = 9;
XPECTED = 9;


 equal")

ED, ACTUAL);
```

(or object) isn't equal to the actual value (or object), we

od of the `Assertions` class. For example, if we want to compare two `Integer` objects, we have to use this assertion:

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Nested;
3  import org.junit.jupiter.api.Test;
4
5  import static org.junit.jupiter.api.Assertions.assertNotEquals;
6
7  @DisplayName("Writing assertions for objects")
8  class ObjectAssertionTest {
9
10     @Nested
11     @DisplayName("When two objects aren't equal")
12     class WhenTwoObjectsAreNotEqual {
13
14         @Nested
15         @DisplayName("When objects are integers")
16         class WhenObjectsAreIntegers {
17
18             private final Integer ACTUAL = 9;
19             private final Integer EXPECTED = 4;
20
21             @Test
22             @DisplayName("Should not be equal")
23             void shouldNotBeEqual() {
24                 assertNotEquals(EXPECTED, ACTUAL);
25             }
26         }
27     }
28 }
```

sertions for object references.

to the same object, we have to use the `assertSame()` words, we have to use this assertion:

```
yName;
;

.Assertions.assertSame;
```

```
7  @DisplayName("Writing assertions for objects")
8  class ObjectAssertionTest {
9
10     @Nested
11     @DisplayName("When two objects refer to the same object")
12     class WhenTwoObjectsReferToSameObject {
13
14         private final Object ACTUAL = new Object();
15         private final Object EXPECTED = ACTUAL;
16
17         @Test
18         @DisplayName("Should refer to the same object")
19         void shouldReferToSameObject() {
20             assertSame(EXPECTED, ACTUAL);
21         }
22     }
23 }
```

If we want to ensure that two objects don't refer to the same object, we have to use the `assertNotSame()` method of the `Assertions` class. In other words, we have to use this assertion:

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Nested;
3  import org.junit.jupiter.api.Test;
4
5  import static org.junit.jupiter.api.Assertions.assertNotSame;
6
7  @DisplayName("Writing assertions for objects")
8  class ObjectAssertionTest {
```

```
9
```

```
don't refer to the same object")
oSameObject {

 = new Object();
ED = new Object();


fer to the same object")
ject() {
 ACTUAL);
```

ify that two arrays are equal.

Asserting That Two Arrays Are Equal

If we want to verify that two arrays are equal, we have to use the `assertArrayEquals()` method of the `Assertions` class. For example, if we want verify that two `int` arrays are equal, we have to use this assertion:

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Nested;
3  import org.junit.jupiter.api.Test;
4
5  import static org.junit.jupiter.api.Assertions.assertArrayEquals;
6
7  @DisplayName("Write assertions for arrays")
8  class ArrayAssertionTest {
9
10     @Nested
11     @DisplayName("When arrays contain integers")
12     class WhenArraysContainIntegers {
13
14         final int[] ACTUAL = new int[]{2, 5, 7};
15         final int[] EXPECTED = new int[]{2, 5, 7};
16
17         @Test
18         @DisplayName("Should contain the same integers")
19         void shouldContainSameIntegers() {
20             assertArrayEquals(EXPECTED, ACTUAL);
```

```
21          }
```


ojects or values. To be more specific, JUnit 5 iterates both ensures that the elements found from the given index are equal.

Next, we will find out how wen can verify that two iterables are equal.

Asserting That Two Iterables Are Equal

If we want to verify that two iterables are deeply equal, we have to use the `assertIterableEquals()` method of the `Assertions` class. For example, if we want verify that two `Integer` lists are deeply equal, we have to use this assertion:

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Nested;
3  import org.junit.jupiter.api.Test;
4
5  import java.util.Arrays;
6  import java.util.List;
7
8  import static org.junit.jupiter.api.Assertions.assertIterableEquals;
9
10 @DisplayName("Writing assertions for lists")
11 class ListAssertionTest {
```

```
wo lists")


 FIRST = Arrays.asList(1, 2, 3);
 SECOND = Arrays.asList(1, 2, 3);


n the same elements")
nts() {
RST, SECOND);
```

Two iterables are considered as equal if:

- They are both `null` or empty.
- Both iterables contain the “same” objects or values. To be more specific, JUnit 5 iterates both iterables one element at a time and ensures that the elements found from the given index are equal.

Let’s move on and find out how we can write assertions for the exception thrown by the system under test.

Writing Assertions for Exceptions

If we want to write assertions for the exceptions thrown by the system under test, we have to use the `assertThrows()` method of the `Assertions` class. This method takes the following method

of the expected exception.

e system under test.

ystem under test throws a `NullPointerException`, our

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Test;
3
4  import static org.junit.jupiter.api.Assertions.assertThrows;
5
6  @DisplayName("Writing assertions for exceptions")
7  class ExceptionAssertionTest {
8
9      @Test
10     @DisplayName("Should throw the correct exception")
11     void shouldThrowCorrectException() {
12         assertThrows(
13                 NullPointerException.class,
14                 () -> { throw new NullPointerException(); }
15         );
16     }
17 }
```

Because the `assertThrows()` method returns the thrown exception object, we can also write additional assertions for the thrown exception. For example, if we want to verify that the thrown exception has the correct message, we can use the following assertions:

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Test;
3
4  import static org.junit.jupiter.api.Assertions.assertEquals;
5  import static org.junit.jupiter.api.Assertions.assertThrows;
```

```
6
```

```
r exceptions")
```

```
xception that has the correct message")
CorrectMessage() {
thrown = assertThrows(
n.class,
ullPointerException("Hello World!"); }

, thrown.getMessage());
```

hod fails if the system under test throws an exception, sometimes we want to explicitly assert that no exception is thrown by the tested code. If this is the case, we have to use the `assertDoesNotThrow()` method of the `Assertions` class. When we want to verify that no exception is thrown by the system under test, we can use one of these two options:

First, if we don't want to write assertions for the value returned by the system under test, we have to pass the following method parameters to the `assertDoesNotThrow()` method:

- An `Executable` object that invokes the system under test.
- An optional error message.

For example, if we want to verify that the system under test doesn't throw an exception, our assertion looks as follows:

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Test;
3
4  import static org.junit.jupiter.api.Assertions.assertDoesNotThrow;
5
6  @DisplayName("Writing assertions for exceptions")
7  class ExceptionAssertionTest {
```

```
8
```

```
an exception")
{
);
```

the value returned by the system under test, we have to pass

`sertDoesNotThrow()` method:

invokes the system under test (and returns the return value).

- An optional error message.

For example, if we want to verify that the system under test doesn't throw an exception AND we want to verify that the system under test returns the correct message, our assertion looks as follows:

```
1  import org.junit.jupiter.api.DisplayName;
2  import org.junit.jupiter.api.Test;
3
4  import static org.junit.jupiter.api.Assertions.assertDoesNotThrow;
5  import static org.junit.jupiter.api.Assertions.assertEquals;
6
7  @DisplayName("Writing assertions for exceptions")
8  class ExceptionAssertionTest {
9
10     @Test
11     @DisplayName("Should not throw an exception")
12     void shouldNotThrowException() {
13         String message = assertDoesNotThrow(() -> { return "Hello World!"; } );
14         assertEquals("Hello World!", message);
15     }
16 }
```

Next, we will find out how we can write assertions for the execution time of the system under test.

Writing Assertions for the Execution Time of the System Under Test

he system under test is completed before a specified timeout

ut()` and `assertTimeoutPreemptively()` methods of the

take the following method parameters:

imeout.

er` object that invokes the system under test.

wn if the specified timeout is exceeded.

Even though these two methods are quite similar, they have one crucial difference. This difference is explained in the following:

- If we use the `assertTimeout()` method, the provided `Executable` or `ThrowingSupplier` will be executed in the same thread as the code that calls it. Also, this method doesn't abort the execution if the timeout is exceeded.
- If we use the `assertTimeoutPreemptively()` method, the provided `Executable` or `ThrowingSupplier` will be executed in a different thread than the code that calls it. Also, this method aborts the execution if the timeout is exceeded.

As we see, we can verify that the execution of the system under test is completed before a specified timeout is exceeded by using one of these two options:

First, if we want that the execution is not aborted if the timeout is exceeded, we have to use the `assertTimeout()` method of the `Assertions` class. For example, if we want to verify that the system

ld!' before the specified timeout (50ms) is exceeded, we

s:

```
yName;


.Assertions.assertEquals;
.Assertions.assertTimeout;

r the execution time of the system under test")
```

```
13      @DisplayName("Should return the correct message before timeout is exceeded")
14      void shouldReturnCorrectMessageBeforeTimeoutIsExceeded() {
15          final String message = assertTimeout(Duration.ofMillis(50), () -> {
16              Thread.sleep(20);
17              return "Hello World!";
18          });
19          assertEquals("Hello World!", message);
20      }
21  }
```

Second, if we want that the execution is aborted if the timeout is exceeded, we have to use the `assertTimeoutPreemptively()` method of the `Assertions` class. For example, if we want to verify that the system under test returns the message: 'Hello world!' before the specified timeout (50ms) is exceeded, we have to write assertions that look as follows:

```
1   import org.junit.jupiter.api.DisplayName;
2   import org.junit.jupiter.api.Test;
3
4   import java.time.Duration;
5
6   import static org.junit.jupiter.api.Assertions.assertEquals;
7   import static org.junit.jupiter.api.Assertions.assertTimeoutPreemptively;
8
9   @DisplayName("Writing assertions for the execution time of the system under test")
10  class TimeoutAssertionTest {
11
12      @Test
13      @DisplayName("Should return the correct message before timeout is exceeded")
```

```
14      void shouldReturnCorrectMessageBeforeTimeoutIsExceeded() {
                                        rtTimeoutPreemptively(Duration.ofMillis(50), () -> {
```



```
                                        , message);
```

```
                                        ss
```

We can now write basic assertions with JUnit 5. Let's move on and find out how we can customize the error messages which are shown by JUnit 5 if an assertion fails.

Providing a Custom Error Message

As we remember, if we want to specify a custom error message that is shown when our assertion fails, we have to pass this message as the last method parameter of the invoked assertion method. We can create this message by using one of these two options:

First, we can create a new `String` object and pass this object as the last method parameter of the invoked assertion assertion method. This is a good choice if our error message has no parameters. For example, if we want to provide a custom error message for an assertion which verifies that a `boolean` value is `false`, we have to write an assertion that looks as follows:

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

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



```
.Assertions.assertFalse;

booleans")


alse")


se")
 boolean is not false");
```

```
}
```

Second, we can create a message supplier (`Supplier<String>`) and pass this supplier as the last method parameter of the invoked assertion method. If we use this approach, JUnit 5 creates the actual error messages **only** if our assertion fails. That's why this is a good choice if we want to create a "complex" error message that has parameters.

For example, if we want to provide a custom error message for an assertion which verifies that a map contains the given key, we have to write an assertion that looks as follows:

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

import java.util.HashMap;
import java.util.Map;

import static org.junit.jupiter.api.Assertions.assertTrue;

@DisplayName("Writing assertions for maps")
class MapAssertionTest {

    private static final String KEY = "key";
    private static final String VALUE = "value";

```

```
16      private Map<String, String> map;
```



```
e correct key")
{

),
("The map doesn't contain the key: %s", KEY)
```

It's good to understand that the custom error message doesn't override the default error message shown if an assertion fails. It is simply a prefix that is prepended to the default error message of the used assertion object. At first, this feels a bit weird, but it's actually quite useful after you get used to it.

Next, we will find out how we can group assertions with JUnit 5.

Grouping Assertions

If we have to write an assertion for a state that requires multiple assertions, we can group our assertions by using the `assertAll()` method of the `Assertions` class. This method takes the following method parameters:

- An optional heading that identifies the asserted state.

of `Executable` objects which invoke our assertions.

l, it invokes all assertions given as a method parameter and

ions have been run.

ertion which verifies that a `Person` object has the correct

s looks as follows:

```
    private String firstName;
    private String lastName;

    public Person() {}

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

As we can see, if we want to verify that a person has the correct name, we have to verify that the asserted `Person` object has the correct first and last name. In other words, we have to write an assertion that looks as follows:

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

```
4
                                    .Assertions.assertAll;
                                    .Assertions.assertEquals;

                                    ions")


                                    ST_NAME = "Jane";
                                    T_NAME = "Doe";




                                    AME);
                                    E);
```

```
22
23      @Test
24      @DisplayName("Should have the correct name")
25      void shouldHaveCorrectName() {
26          assertAll("name",
27                  () -> assertEquals(FIRST_NAME,
28                          person.getFirstName(),
29                          "The first name is incorrect"
30                  ),
31                  () -> assertEquals(LAST_NAME,
32                          person.getLastName(),
33                          "The last name is incorrect"
34                  )
35          );
36      }
37  }
```

We can now write basic assertions with JUnit 5, provide a custom error message that is shown when an assertion fail, and group assertions with JUnit 5.

Additional Reading:

- JUnit 5 User Guide: 2.4 Assertions

~~class~~

s blog post.

~~This blog post has taught us four things:~~

- If we want to write assertions by using the "standard" JUnit 5 API, we must use the `org.junit.jupiter.api.Assertions` class.
- If we want to specify a custom error message that has no parameters, we have to create a new `String` object and pass this object as the last method parameter of the invoked assertion method.
- If we want to specify a "complex" error message that has parameters, we have to create a message supplier (`Supplier<String>`) and pass this supplier as the last method parameter of the invoked assertion method.
- If we have to write an assertion for a state that requires multiple assertions, we can group our assertions by using the `assertAll()` method of the `Assertions` class.

P.S. You can get the example application of this blog post from Github.

RELATED POSTS

WEEKLY /

Java Testing Weekly 30 / 2017

WEEKLY /

Java Testing Weekly 8 / 2019

WEEKLY /

April 17, 2018, 19:03

Hello,
Thanks for this nice article. It really help me understand assertions.

I have one query,
In case of failed group assertion, is it possible to execute next statement

…
…
assertAll(“test_Date Assertion”,
() -> assertEquals(CommUtils.getXpathField(xmlSoapResponse, “responseCode”),”01″),
() -> assertTrue(this.assertionUtility.isValidSOAPResponse(xmlSoapResponse)),
() -> assertEquals(datesFromDB.getCurrentProcessingDate().toString(), processFromService));
log.debug(“Second Set”);

Try to use a Thread between every statement or make loop

REPLY

Ardalan Link

December 4, 2018, 15:40

nice Article Prefect,Merci

REPLY

Asafir Link

January 8, 2019, 10:18

ertThrow in Groovy a RunTimeException is thrown:

d org.junit.jupiter.api.Assertions#assertTimeout

- [class java.time.Duration]

it.jupiter.api.function.ThrowingSupplier]

it.jupiter.api.function.Executable]

Many thanks!

REPLY

Petri Link

January 8, 2019, 19:00

Hi,

Have you tried to specify the type of your method parameter by using the as keyword?

REPLY

Leave a Comment

PREVIOUS POST: JAVA TESTING WEEKLY 11 / 2018

NEXT POST: JAVA TESTING WEEKLY 12 / 2018

Writing Tests for Data Access Code

HOW TO WRITE BETTER TESTS?

If you are struggling to write automated tests that **embrace change**, you should find out how my testing course can help you to **write tests for Spring and Spring Boot applications**.

Spring Social Tutorial

Using jOOQ with Spring

BUILD YOUR APPLICATION

Getting Started With Gradle

Maven Tutorial

FIND THE BEST TUTORIALS

JUnit 5 - The Ultimate Resource

Spring Batch - The Ultimate Resource

SEARCH

rollers: Configuration

otlin and JUnit 5: Configuration

The Best Way to Configure the Spring MVC Test Framework, Part One

TestProject Best Practices

Writing Custom Web Element Actions With TestProject

HOW TO WRITE BETTER TESTS?

If you are struggling to write automated tests that **embrace change**, you should find out how my testing course can help you to **write tests for Spring and Spring Boot applications**.