**JUnit 5 Assertions Examples**

**JUnit 5 assertions** help in validating the expected output with actual output of a testcase. To keep things simple, all **JUnit Jupiter assertions** are static methods in the [org.junit.jupiter.Assertions](https://junit.org/junit5/docs/current/api/org/junit/jupiter/api/Assertions.html) class.

Table of Contents

[Assertions.assertEquals() and Assertions.assertNotEquals()](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#assertEquals-assertNotEquals)

[Assertions.assertArrayEquals()](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#assertArrayEquals)

[Assertions.assertIterableEquals()](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#assertIterableEquals)

[Assertions.assertLinesMatch()](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#assertLinesMatch)

[Assertions.assertNotNull() and Assertions.assertNull()](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#assertNotNull-assertNull)

[Assertions.assertNotSame() and Assertions.assertSame()](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#assertNotSame-assertSame)

[Assertions.assertTimeout() and Assertions.assertTimeoutPreemptively()](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#assertTimeout-assertTimeoutPreemptively)

[Assertions.assertTrue() and Assertions.assertFalse()](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#assertTrue-assertFalse)

[Assertions.assertThrows()](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#assertThrows)

[Assertions.fail()](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#fail)

**Assertions.assertEquals() and Assertions.assertNotEquals() Example**

Use Assertions.assertEquals() to assert that **expected value and actual value are equal**. assertEquals() has many overloaded methods for different data types e.g. int, short, float, char etc. It also support passing error message to be printed in case test fails. e.g.

public static void assertEquals(int expected, int actual)

public static void assertEquals(int expected, int actual, String message)

public static void assertEquals(int expected, int actual, Supplier<String< messageSupplier)

|  |
| --- |
| void testCase()  {      //Test will pass      Assertions.assertEquals(4, Calculator.add(2, 2));        //Test will fail      Assertions.assertEquals(3, Calculator.add(2, 2), "Calculator.add(2, 2) test failed");        //Test will fail      Supplier&lt;String&gt; messageSupplier  = ()-> "Calculator.add(2, 2) test failed";      Assertions.assertEquals(3, Calculator.add(2, 2), messageSupplier);  } |

Similarly, Assertions.assertNotEquals() method is used to assert that **expected value and actual value are NOT equal**. In contrast to assertEquals(), assertNotEquals() does not overloaded methods for different data types but only Object is accepted.

public static void assertNotEquals(Object expected, Object actual)

public static void assertNotEquals(Object expected, Object actual, String message)

public static void assertNotEquals(Object expected, Object actual, Supplier<String> messageSupplier)

|  |
| --- |
| void testCase()  {      //Test will pass      Assertions.assertNotEquals(3, Calculator.add(2, 2));        //Test will fail      Assertions.assertNotEquals(4, Calculator.add(2, 2), "Calculator.add(2, 2) test failed");        //Test will fail      Supplier&lt;String&gt; messageSupplier  = ()-> "Calculator.add(2, 2) test failed";      Assertions.assertNotEquals(4, Calculator.add(2, 2), messageSupplier);  } |

**Assertions.assertArrayEquals() Example**

Similar to assertEquals(), assertArrayEquals() does the same for arrays i.e. asserts that **expected and actual arrays are equal**. It also has overloaded methods for different data types e.g. boolean[], char[], int[] etc. It also support passing error message to be printed in case test fails. e.g.

public static void assertArrayEquals(int[] expected, int[] actual)

public static void assertArrayEquals(int[] expected, int[] actual, String message)

public static void assertArrayEquals(int[] expected, int[] actual, Supplier<String> messageSupplier)

|  |
| --- |
| void testCase()  {      //Test will pass      Assertions.assertArrayEquals(new int[]{1,2,3}, new int[]{1,2,3}, "Array Equal Test");        //Test will fail because element order is different      Assertions.assertArrayEquals(new int[]{1,2,3}, new int[]{1,3,2}, "Array Equal Test");        //Test will fail because number of elements are different      Assertions.assertArrayEquals(new int[]{1,2,3}, new int[]{1,2,3,4}, "Array Equal Test");  } |

**Assertions.assertIterableEquals() Example**

It asserts that **expected and actual iterables are deeply equal**. Deeply equal means that number and order of elements in collection must be same; as well as iterated elements must be equal.

It also has 3 overloaded methods.

public static void assertIterableEquals(Iterable<?> expected, Iterable> actual)

public static void assertIterableEquals(Iterable<?> expected, Iterable> actual, String message)

public static void assertIterableEquals(Iterable<?> expected, Iterable> actual, Supplier<String> messageSupplier)

|  |
| --- |
| @Test  void testCase()  {       Iterable<Integer> listOne = new ArrayList<>(Arrays.asList(1,2,3,4));       Iterable<Integer> listTwo = new ArrayList<>(Arrays.asList(1,2,3,4));       Iterable<Integer> listThree = new ArrayList<>(Arrays.asList(1,2,3));       Iterable<Integer> listFour = new ArrayList<>(Arrays.asList(1,2,4,3));        //Test will pass      Assertions.assertIterableEquals(listOne, listTwo);        //Test will fail      Assertions.assertIterableEquals(listOne, listThree);        //Test will fail      Assertions.assertIterableEquals(listOne, listFour);  } |

**Assertions.assertLinesMatch() Example**

It asserts that **expected list of Strings matches actual list**. The logic to match a string with another string is :

1. check if expected.equals(actual) – if yes, continue with next pair
2. otherwise treat expected as a regular expression and check via  
   [String.matches(String)](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#matches-java.lang.String-) – if yes, continue with next pair
3. otherwise check if expected line is a fast-forward marker, if yes apply  
   fast-forward actual lines accordingly and goto 1.

A valid fast-forward marker is string which start and end with >> and and contains at least 4 characters. Any character between the fast-forward literals are discarded.

>>>>

>> stacktrace >>

>> single line, non Integer.parse()-able comment >>

**Assertions.assertNotNull() and Assertions.assertNull() Example**

assertNotNull() asserts that **actual is NOT null**. Similarly, assertNull() method asserts that **actual is null**. Both has three overloaded methods.

public static void assertNotNull(Object actual)

public static void assertNotNull(Object actual, String message)

public static void assertNotNull(Object actual, Supplier<String> messageSupplier)

public static void assertEquals(Object actual)

public static void assertEquals(Object actual, String message)

public static void assertEquals(Object actual, Supplier<String> messageSupplier)

|  |
| --- |
| @Test  void testCase()  {      String nullString = null;      String notNullString = "howtodoinjava.com";        //Test will pass      Assertions.assertNotNull(notNullString);        //Test will fail      Assertions.assertNotNull(nullString);        //Test will pass      Assertions.assertNull(nullString);        // Test will fail      Assertions.assertNull(notNullString);  } |

**Assertions.assertNotSame() and Assertions.assertSame() Example**

assertNotSame() asserts that **expected and actual DO NOT refer to the same object.**. Similarly, assertSame() method asserts that **expected and actual refer to exactly same object.**. Both has three overloaded methods.

public static void assertNotSame(Object actual)

public static void assertNotSame(Object actual, String message)

public static void assertNotSame(Object actual, Supplier<> messageSupplier)

public static void assertSame(Object actual)

public static void assertSame(Object actual, String message)

public static void assertSame(Object actual, Supplier<String> messageSupplier)

|  |
| --- |
| @Test  void testCase()  {      String originalObject = "howtodoinjava.com";      String cloneObject = originalObject;      String otherObject = "example.com";        //Test will pass      Assertions.assertNotSame(originalObject, otherObject);        //Test will fail      Assertions.assertNotSame(originalObject, cloneObject);        //Test will pass      Assertions.assertSame(originalObject, cloneObject);        // Test will fail      Assertions.assertSame(originalObject, otherObject);  } |

**Assertions.assertTimeout() and Assertions.assertTimeoutPreemptively() Example**

assertTimeout() and assertTimeoutPreemptively() both are used to test long running tasks. If given task inside testcase takes more than specified duration, then test will fail.

Only different between both methods is that in assertTimeoutPreemptively(), execution of the Executable or ThrowingSupplier will be preemptively aborted if the timeout is exceeded. In case of assertTimeout(), Executable or ThrowingSupplier will NOT be aborted.

public static void assertTimeout(Duration timeout, Executable executable)

public static void assertTimeout(Duration timeout, Executable executable, String message)

public static void assertTimeout(Duration timeout, Executable executable, Supplier<String> messageSupplier)

public static void assertTimeout(Duration timeout, ThrowingSupplier<T> supplier, String message)

public static void assertTimeout(Duration timeout, ThrowingSupplier<T> supplier, Supplier<String> messageSupplier)

|  |
| --- |
| @Test  void testCase() {        //This will pass      Assertions.assertTimeout(Duration.ofMinutes(1), () -> {          return "result";      });        //This will fail      Assertions.assertTimeout(Duration.ofMillis(100), () -> {          Thread.sleep(200);          return "result";      });        //This will fail      Assertions.assertTimeoutPreemptively(Duration.ofMillis(100), () -> {          Thread.sleep(200);          return "result";      });  } |

**Assertions.assertTrue() and Assertions.assertFalse() Example**

assertTrue() asserts that the supplied condition is true or boolean condition supplied by BooleanSupplier is true. Similarly, assertFalse() asserts that **supplied condition is false**. It has following overloaded methods:

public static void assertTrue(boolean condition)

public static void assertTrue(boolean condition, String message)

public static void assertTrue(boolean condition, Supplier<String> messageSupplier)

public static void assertTrue(BooleanSupplier booleanSupplier)

public static void assertTrue(BooleanSupplier booleanSupplier, String message)

public static void assertTrue(BooleanSupplier booleanSupplier, Supplier<String> messageSupplier)

public static void assertFalse(boolean condition)

public static void assertFalse(boolean condition, String message)

public static void assertFalse(boolean condition, Supplier<String> messageSupplier)

public static void assertFalse(BooleanSupplier booleanSupplier)

public static void assertFalse(BooleanSupplier booleanSupplier, String message)

public static void assertFalse(BooleanSupplier booleanSupplier, Supplier<String> messageSupplier)

|  |
| --- |
| @Test  void testCase() {        boolean trueBool = true;      boolean falseBool = false;        Assertions.assertTrue(trueBool);      Assertions.assertTrue(falseBool, "test execution message");      Assertions.assertTrue(falseBool, AppTest::message);      Assertions.assertTrue(AppTest::getResult, AppTest::message);        Assertions.assertFalse(falseBool);      Assertions.assertFalse(trueBool, "test execution message");      Assertions.assertFalse(trueBool, AppTest::message);      Assertions.assertFalse(AppTest::getResult, AppTest::message);  }    private static String message () {      return "Test execution result";  }    private static boolean getResult () {      return true;  } |

**Assertions.assertThrows() Example**

It asserts that execution of the supplied Executable throws an exception of the expectedType and returns the exception.

public static <T extends Throwable> T assertThrows(Class<T> expectedType,

Executable executable)

|  |
| --- |
| @Test  void testCase() {        Throwable exception = Assertions.assertThrows(IllegalArgumentException.class, () -> {          throw new IllegalArgumentException("error message");      });  } |

**Assertions.fail() Example**

fail() method simply fails the test. It has following overloaded methods:

public static void fail(String message)

public static void fail(Throwable cause)

public static void fail(String message, Throwable cause)

public static void fail(Supplier<String> messageSupplier)

|  |
| --- |
| public class AppTest {      @Test      void testCase() {            Assertions.fail("not found good reason to pass");          Assertions.fail(AppTest::message);      }        private static String message () {          return "not found good reason to pass";      }  } |