**Junit**

[**https://junit.org/junit4/**](https://junit.org/junit4/)

**Assertions**

Son un conjunto de métodos que nos permite definir los resultados esperados de una prueba de unidad y dependiendo si el resultado esperado es correcto la prueba se define como exitosa en caso contrario es fallida.

|  |
| --- |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(boolean[], boolean[]))**(boolean[] expecteds, boolean[] actuals)  Asserts that two boolean arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(byte[], byte[]))**(byte[] expecteds, byte[] actuals) Asserts that two byte arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(char[], char[]))**(char[] expecteds, char[] actuals) Asserts that two char arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(double[], double[], double))**(double[] expecteds, double[] actuals, double delta) Asserts that two double arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(float[], float[], float))**(float[] expecteds, float[] actuals, float delta) Asserts that two float arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(int[], int[]))**(int[] expecteds, int[] actuals) Asserts that two int arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(long[], long[]))**(long[] expecteds, long[] actuals) Asserts that two long arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(java.lang.Object[], java.lang.Object[]))**([Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html)[] expecteds, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html)[] actuals) Asserts that two object arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(short[], short[]))**(short[] expecteds, short[] actuals) Asserts that two short arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(java.lang.String, boolean[], boolean[]))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, boolean[] expecteds, boolean[] actuals) Asserts that two boolean arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(java.lang.String, byte[], byte[]))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, byte[] expecteds, byte[] actuals) Asserts that two byte arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(java.lang.String, char[], char[]))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, char[] expecteds, char[] actuals) Asserts that two char arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(java.lang.String, double[], double[], double))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, double[] expecteds, double[] actuals, double delta) Asserts that two double arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(java.lang.String, float[], float[], float))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, float[] expecteds, float[] actuals, float delta) Asserts that two float arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(java.lang.String, int[], int[]))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, int[] expecteds, int[] actuals) Asserts that two int arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(java.lang.String, long[], long[]))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, long[] expecteds, long[] actuals) Asserts that two long arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(java.lang.String, java.lang.Object[], java.lang.Object[]))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html)[] expecteds, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html)[] actuals) Asserts that two object arrays are equal. |
| **[assertArrayEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertArrayEquals(java.lang.String, short[], short[]))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, short[] expecteds, short[] actuals) Asserts that two short arrays are equal. |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(double, double))**(double expected, double actual) **Deprecated.** *Use assertEquals(double expected, double actual, double delta) instead* |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(double, double, double))**(double expected, double actual, double delta) Asserts that two doubles are equal to within a positive delta. |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(float, float, float))**(float expected, float actual, float delta) Asserts that two floats are equal to within a positive delta. |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(long, long))**(long expected, long actual) Asserts that two longs are equal. |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(java.lang.Object[], java.lang.Object[]))**([Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html)[] expecteds,[Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html)[] actuals) **Deprecated.** *use assertArrayEquals* |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(java.lang.Object, java.lang.Object))**([Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) expected, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) actual) Asserts that two objects are equal. |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(java.lang.String, double, double))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, double expected, double actual) **Deprecated.** *Use assertEquals(String message, double expected, double actual, double delta) instead* |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(java.lang.String, double, double, double))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, double expected, double actual, double delta) Asserts that two doubles are equal to within a positive delta. |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(java.lang.String, float, float, float))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, float expected, float actual, float delta) Asserts that two floats are equal to within a positive delta. |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(java.lang.String, long, long))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, long expected, long actual) Asserts that two longs are equal. |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(java.lang.String, java.lang.Object[], java.lang.Object[]))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html)[] expecteds, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html)[] actuals)  **Deprecated.** *use assertArrayEquals* |
| **[assertEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertEquals(java.lang.String, java.lang.Object, java.lang.Object))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) expected, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) actual) Asserts that two objects are equal. |
| **[assertFalse](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertFalse(boolean))**(boolean condition) Asserts that a condition is false. |
| **[assertFalse](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertFalse(java.lang.String, boolean))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, boolean condition) Asserts that a condition is false. |
| **[assertNotEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotEquals(double, double, double))**(double unexpected, double actual, double delta) Asserts that two doubles are **not** equal to within a positive delta. |
| **[assertNotEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotEquals(float, float, float))**(float unexpected, float actual, float delta) Asserts that two floats are **not** equal to within a positive delta. |
| **[assertNotEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotEquals(long, long))**(long unexpected, long actual) Asserts that two longs are **not** equals. |
| **[assertNotEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotEquals(java.lang.Object, java.lang.Object))**([Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) unexpected, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) actual) Asserts that two objects are **not** equals. |
| **[assertNotEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotEquals(java.lang.String, double, double, double))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, double unexpected, double actual, double delta) Asserts that two doubles are **not** equal to within a positive delta. |
| **[assertNotEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotEquals(java.lang.String, float, float, float))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, float unexpected, float actual, float delta) Asserts that two floats are **not** equal to within a positive delta. |
| **[assertNotEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotEquals(java.lang.String, long, long))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, long unexpected, long actual)  Asserts that two longs are **not e**quals. |
| **[assertNotEquals](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotEquals(java.lang.String, java.lang.Object, java.lang.Object))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) unexpected, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) actual) Asserts that two objects are **not** equals. |
| **[assertNotNull](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotNull(java.lang.Object))**([Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) object)  Asserts that an object isn't null. |
| **[assertNotNull](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotNull(java.lang.String, java.lang.Object))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) object) Asserts that an object isn't null. |
| **[assertNotSame](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotSame(java.lang.Object, java.lang.Object))**([Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) unexpected, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) actual) Asserts that two objects do not refer to the same object. |
| **[assertNotSame](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNotSame(java.lang.String, java.lang.Object, java.lang.Object))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) unexpected, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) actual) Asserts that two objects do not refer to the same object. |
| **[assertNull](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNull(java.lang.Object))**([Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) object) Asserts that an object is null. |
| **[assertNull](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertNull(java.lang.String, java.lang.Object))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) object) Asserts that an object is null. |
| **[assertSame](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertSame(java.lang.Object, java.lang.Object))**([Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) expected, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) actual) Asserts that two objects refer to the same object. |
| **[assertSame](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertSame(java.lang.String, java.lang.Object, java.lang.Object))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) expected, [Object](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html) actual) Asserts that two objects refer to the same object. |
| **[assertThat](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertThat(java.lang.String, T, org.hamcrest.Matcher))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) reason, T actual, [Matcher](https://junit.org/junit4/javadoc/4.12/org/hamcrest/Matcher.html)<? super T> matcher) Asserts that actual satisfies the condition specified by matcher. |
| **[assertThat](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertThat(T, org.hamcrest.Matcher))**(T actual, [Matcher](https://junit.org/junit4/javadoc/4.12/org/hamcrest/Matcher.html)<? super T> matcher) Asserts that actual satisfies the condition specified by matcher. |
| **[assertTrue](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertTrue(boolean))**(boolean condition) Asserts that a condition is true. |
| **[assertTrue](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "assertTrue(java.lang.String, boolean))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message, boolean condition) Asserts that a condition is true. |
| **[fail](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "fail())**() Fails a test with no message. |
| **[fail](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html" \l "fail(java.lang.String))**([String](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/String.html) message) Fails a test with the given message. |

**Test Runners**

JUnit proporciona herramientas para definir la suite que se ejecutará y mostrar sus resultados. Para ejecutar pruebas y ver los resultados en la consola, ejecute esto desde un programa Java:

org.junit.runner.JUnitCore.runClasses(TestClass1.class, ...);

org.junit.runner.JUnitCore TestClass1 [...other test classes…]

**Anotaciones**

|  |
| --- |
| **@Test**  The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case. |
| **@Before**  Several tests need similar objects created before they can run. Annotating a public void method with @Before causes that method to be run before each Test method. |
| **@After**  If you allocate external resources in a Before method, you need to release them after the test runs. Annotating a public void method with @After causes that method to be run after the Test method. |
| **@BeforeClass**  Annotating a public static void method with @BeforeClass causes it to be run once before any of the test methods in the class. |
| **@AfterClass**  This will perform the method after all tests have finished. This can be used to perform clean-up activities. |
| **@Ignore**  The Ignore annotation is used to ignore the test and that test will not be executed. |

**Test Run**

@RunWith(JUnit4.class)

public class FeatureTestSuite {

// the class remains empty,

// used only as a holder for the above annotations

}

Notación para definir a una clase auto arranca ble.

**Test Suite**

@RunWith(Suite.class)

@Suite.SuiteClasses({

TestFeatureLogin.class,

TestFeatureLogout.class,

TestFeatureNavigate.class,

TestFeatureUpdate.class

})

public class FeatureTestSuite {

// the class remains empty,

// used only as a holder for the above annotations

}

Notación para definir a una conjunto de suite de pruebas agrupado

**Test Sort**

@FixMethodOrder(MethodSorters.NAME\_ASCENDING)

public class TestMethodOrder {

@Test

public void testA() {

System.out.println("first");

}

@Test

public void testB() {

System.out.println("second");

}

@Test

public void testC() {

System.out.println("third");

}

}

Notación para ordenar las ejecución de pruebas

|  |
| --- |
| **[DEFAULT](https://junit.org/junit4/javadoc/latest/org/junit/runners/MethodSorters.html" \l "DEFAULT)** Sorts the test methods in a deterministic, but not predictable, order |
| **[JVM](https://junit.org/junit4/javadoc/latest/org/junit/runners/MethodSorters.html" \l "JVM)** Leaves the test methods in the order returned by the JVM. |
| **[NAME\_ASCENDING](https://junit.org/junit4/javadoc/latest/org/junit/runners/MethodSorters.html" \l "NAME_ASCENDING)** Sorts the test methods by the method name, in lexicographic order, with [Method.toString()](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/reflect/Method.html?is-external=true" \l "toString()) used as a tiebreaker |

**Test Exception**

**@Test(expected = IndexOutOfBoundsException.class)**

public void empty() {

new ArrayList<Object>().get(0);

}

**@Test**

public void testExceptionMessage() {

try {

new ArrayList<Object>().get(0);

fail("Expected an IndexOutOfBoundsException to be thrown");

} catch (IndexOutOfBoundsException anIndexOutOfBoundsException) {

assertThat(anIndexOutOfBoundsException.getMessage(), is("Index: 0, Size: 0"));

}

}

Notaciones para ignorar Exception en las pruebas

**ExpectedException Rule**

Alternativamente, use la regla ExpectedException. Esta regla le permite indicar no solo qué excepción espera, sino también el mensaje de excepción que espera:

@Rule

public ExpectedException thrown = ExpectedException.none();

@Test

public void shouldTestExceptionMessage() throws IndexOutOfBoundsException {

List<Object> list = new ArrayList<Object>();

thrown.expect(IndexOutOfBoundsException.class);

thrown.expectMessage("Index: 0, Size: 0");

list.get(0); // execution will never get past this line

}

Se puede utilizar Match para validar la exception: thrown.expectMessage(CoreMatchers.containsString("Size: 0"));

public class TestExy {

@Rule

public ExpectedException thrown = ExpectedException.none();

@Test

public void shouldThrow() {

TestThing testThing = new TestThing();

thrown.expect(NotFoundException.class);

thrown.expectMessage(startsWith("some Message"));

thrown.expect(hasProperty("response", hasProperty("status", is(404))));

testThing.chuck();

}

private class TestThing {

public void chuck() {

Response response = Response.status(Status.NOT\_FOUND).entity("Resource not found").build();

throw new NotFoundException("some Message", response);

}

}

}

**assertThat**

assertThat(x, is(3));

assertThat(x, is(not(4)));

assertThat(responseString, either(containsString("color")).or(containsString("colour")));

assertThat(myList, hasItem("3"));

Notación

assertThat([value], [matcher statement]);

not(s)

either(s).or(t)

each(s)

afterFiveSeconds(s)

assertTrue(responseString.contains("color") || responseString.contains("colour"));

// ==> failure message:

// java.lang.AssertionError:

assertThat(responseString, anyOf(containsString("color"), containsString("colour")));

// ==> failure message:

// java.lang.AssertionError:

// Expected: (a string containing "color" or a string containing "colour")

// got: "Please choose a font"

**Core**

* + anything - always matches, useful if you don't care what the object under test is
  + describedAs - decorator to adding custom failure description
  + is - decorator to improve readability - see "Sugar", below
* Logical
  + allOf - matches if all matchers match, short circuits (like Java &&)
  + anyOf - matches if any matchers match, short circuits (like Java ||)
  + not - matches if the wrapped matcher doesn't match and vice versa
* Object
  + equalTo - test object equality using Object.equals
  + hasToString - test Object.toString
  + instanceOf, isCompatibleType - test type
  + notNullValue, nullValue - test for null
  + sameInstance - test object identity
* Beans
  + hasProperty - test JavaBeans properties
* Collections
  + array - test an array's elements against an array of matchers
  + hasEntry, hasKey, hasValue - test a map contains an entry, key or value
  + hasItem, hasItems - test a collection contains elements
  + hasItemInArray - test an array contains an element
* Number
  + closeTo- test floating point values are close to a given value
  + greaterThan, greaterThanOrEqualTo, lessThan, lessThanOrEqualTo - test ordering
* Text
  + equalToIgnoringCase - test string equality ignoring case
  + equalToIgnoringWhiteSpace - test string equality ignoring differences in runs of whitespace
  + containsString, endsWith, startsWith - test string matching

<http://hamcrest.org/JavaHamcrest/javadoc/2.1/>

<https://junit.org/junit4/javadoc/latest/org/junit/matchers/JUnitMatchers.html>

<https://junit.org/junit4/javadoc/latest/org/hamcrest/CoreMatchers.html>

**Ignoring a Test**

@Ignore("Test is ignored as a demonstration")

@Test

public void testSame() {

assertThat(1, is(1));

}

Notación para ignorar una prueba

**Timeout**

@Test(timeout=1000)

public void testWithTimeout() {

...

}

**Timeout Rule**

public class HasGlobalTimeout {

public static String log;

private final CountDownLatch latch = new CountDownLatch(1);

@Rule

public Timeout globalTimeout = Timeout.seconds(10); // 10 seconds max per method tested

@Test

public void testSleepForTooLong() throws Exception {

log += "ran1";

TimeUnit.SECONDS.sleep(100); // sleep for 100 seconds

}

@Test

public void testBlockForever() throws Exception {

log += "ran2";

latch.await(); // will block

}

}

**Parameterized**

@RunWith(Parameterized.class)

public class FibonacciTest {

@Parameters

public static Collection<Object[]> data() {

return Arrays.asList(new Object[][] {

{ 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }, { 4, 3 }, { 5, 5 }, { 6, 8 }

});

}

private int fInput;

private int fExpected;

public FibonacciTest(int input, int expected) {

this.fInput = input;

this.fExpected = expected;

}

@Test

public void test() {

assertEquals(fExpected, Fibonacci.compute(fInput));

}

}

public class Fibonacci {

public static int compute(int n) {

int result = 0;

if (n <= 1) {

result = n;

} else {

result = compute(n - 1) + compute(n - 2);

}

return result;

}

}

**Using @Parameter for Field injection instead of Constructor**

@RunWith(Parameterized.class)

public class FibonacciTest {

@Parameters

public static Collection<Object[]> data() {

return Arrays.asList(new Object[][] {

{ 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }, { 4, 3 }, { 5, 5 }, { 6, 8 }

});

}

@Parameter // first data value (0) is default

public /\* NOT private \*/ int fInput;

@Parameter(1)

public /\* NOT private \*/ int fExpected;

@Test

public void test() {

assertEquals(fExpected, Fibonacci.compute(fInput));

}

}

public class Fibonacci {

...

}

**Tests with single parameter**

@Parameters

public static Iterable<? extends Object> data() {

return Arrays.asList("first test", "second test");

}

@Parameters

public static Object[] data() {

return new Object[] { "first test", "second test" };

}

**Identify Individual test cases**

In order to easily identify the individual test cases in a Parameterized test, you may provide a name using the @Parameters annotation. This name is allowed to contain placeholders that are replaced at runtime:

* {index}: the current parameter index
* {0}, {1}, …: the first, second, and so on, parameter value. NOTE: single quotes ' should be escaped as two single quotes ''.

@RunWith(Parameterized.class)

public class FibonacciTest {

@Parameters(name = "{index}: fib({0})={1}")

public static Iterable<Object[]> data() {

return Arrays.asList(new Object[][] {

{ 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }, { 4, 3 }, { 5, 5 }, { 6, 8 }

});

}

private int input;

private int expected;

public FibonacciTest(int input, int expected) {

this.input = input;

this.expected = expected;

}

@Test

public void test() {

assertEquals(expected, Fibonacci.compute(input));

}

}

public class Fibonacci {

...

}

**Assumptions**

Sin embargo, a veces esto no es deseable o posible. Es bueno poder ejecutar una prueba contra el código tal como está escrito actualmente, suposiciones implícitas y todo, o escribir una prueba que exponga un error conocido. Para estas situaciones, JUnit ahora incluye la capacidad de expresar "suposiciones":

@Test public void filenameIncludesUsername() {

assumeThat(File.separatorChar, is('/'));

assertThat(new User("optimus").configFileName(), is("configfiles/optimus.cfg"));

}

@Test public void correctBehaviorWhenFilenameIsNull() {

assumeTrue(bugFixed("13356")); // bugFixed is not included in JUnit

assertThat(parse(null), is(new NullDocument()));

}

**Rules**

Las reglas permiten una adición o re definición muy flexible del comportamiento de cada método de prueba en una clase de prueba. Los evaluadores pueden reutilizar o ampliar una de las Reglas proporcionadas a continuación, o escribir la suya propia.

public class DigitalAssetManagerTest {

@Rule

public final TemporaryFolder tempFolder = new TemporaryFolder();

@Rule

public final ExpectedException exception = ExpectedException.none();

@Test

public void countsAssets() throws IOException {

File icon = tempFolder.newFile("icon.png");

File assets = tempFolder.newFolder("assets");

createAssets(assets, 3);

DigitalAssetManager dam = new DigitalAssetManager(icon, assets);

assertEquals(3, dam.getAssetCount());

}

private void createAssets(File assets, int numberOfAssets) throws IOException {

for (int index = 0; index < numberOfAssets; index++) {

File asset = new File(assets, String.format("asset-%d.mpg", index));

Assert.assertTrue("Asset couldn't be created.", asset.createNewFile());

}

}

@Test

public void throwsIllegalArgumentExceptionIfIconIsNull() {

exception.expect(IllegalArgumentException.class);

exception.expectMessage("Icon is null, not a file, or doesn't exist.");

new DigitalAssetManager(null, null);

}

}

**TemporaryFolder Rule**

La regla de TemporaryFolder permite la creación de archivos y carpetas que se eliminan cuando finaliza el método de prueba (ya sea que pase o falle). Por defecto, no se produce ninguna excepción si los recursos no se pueden eliminar:

public static class HasTempFolder {

@Rule

public final TemporaryFolder folder = new TemporaryFolder();

@Test

public void testUsingTempFolder() throws IOException {

File createdFile = folder.newFile("myfile.txt");

File createdFolder = folder.newFolder("subfolder");

}

}

@Rule

public TemporaryFolder folder = TemporaryFolder.builder().assureDeletion().build();

**ExternalResource Rules**

ExternalResource es una clase base para Reglas (como TemporaryFolder) que configura un recurso externo antes de una prueba (un archivo, socket, servidor, conexión de base de datos, etc.) y garantiza que lo derribará después:

public static class UsesExternalResource {

Server myServer = new Server();

@Rule

public final ExternalResource resource = new ExternalResource() {

@Override

protected void before() throws Throwable {

myServer.connect();

};

@Override

protected void after() {

myServer.disconnect();

};

};

@Test

public void testFoo() {

new Client().run(myServer);

}

}

**ErrorCollector Rule**

La regla ErrorCollector permite que la ejecución de una prueba continúe después de encontrar el primer problema (por ejemplo, para recopilar todas las filas incorrectas en una tabla e informarlas todas a la vez):

public static class UsesErrorCollectorTwice {

@Rule

public final ErrorCollector collector = new ErrorCollector();

@Test

public void example() {

collector.addError(new Throwable("first thing went wrong"));

collector.addError(new Throwable("second thing went wrong"));

}

}

**Verifier Rule**

Verifier es una clase base para Reglas como ErrorCollector, que puede convertir los métodos de prueba aprobados en pruebas fallidas si falla una verificación.

public static class UsesVerifier {

private static String sequence;

@Rule

public final Verifier collector = new Verifier() {

@Override

protected void verify() {

sequence += "verify ";

}

};

@Test

public void example() {

sequence += "test ";

}

@Test

public void verifierRunsAfterTest() {

sequence = "";

assertThat(testResult(UsesVerifier.class), isSuccessful());

assertEquals("test verify ", sequence);

}

}

**TestWatchman/TestWatcher Rules**

<https://junit.org/junit4/javadoc/latest/org/junit/rules/TestWatcher.html>

public class WatchmanTest {

private static String watchedLog;

@Rule

public final TestRule watchman = new TestWatcher() {

@Override

public Statement apply(Statement base, Description description) {

return super.apply(base, description);

}

@Override

protected void succeeded(Description description) {

watchedLog += description.getDisplayName() + " " + "success!\n";

}

@Override

protected void failed(Throwable e, Description description) {

watchedLog += description.getDisplayName() + " " + e.getClass().getSimpleName() + "\n";

}

@Override

protected void skipped(AssumptionViolatedException e, Description description) {

watchedLog += description.getDisplayName() + " " + e.getClass().getSimpleName() + "\n";

}

@Override

protected void starting(Description description) {

super.starting(description);

}

@Override

protected void finished(Description description) {

super.finished(description);

}

};

@Test

public void fails() {

fail();

}

@Test

public void succeeds() {

}

}

**TestName Rule**

La regla TestName hace que el nombre de la prueba actual esté disponible dentro de los métodos de prueba:

public class NameRuleTest {

@Rule

public final TestName name = new TestName();

@Test

public void testA() {

assertEquals("testA", name.getMethodName());

}

@Test

public void testB() {

assertEquals("testB", name.getMethodName());

}

}

**Timeout Rule**

La regla de tiempo de espera aplica el mismo tiempo de espera a todos los métodos de prueba en una clase:

public static class HasGlobalTimeout {

public static String log;

@Rule

public final TestRule globalTimeout = Timeout.millis(20);

@Test

public void testInfiniteLoop1() {

log += "ran1";

for(;;) {}

}

@Test

public void testInfiniteLoop2() {

log += "ran2";

for(;;) {}

}

}

**ExpectedException Rules**

La regla ExpectedException permite la especificación en prueba de los tipos y mensajes de excepción esperados:

public static class HasExpectedException {

@Rule

public final ExpectedException thrown = ExpectedException.none();

@Test

public void throwsNothing() {

}

@Test

public void throwsNullPointerException() {

thrown.expect(NullPointerException.class);

throw new NullPointerException();

}

@Test

public void throwsNullPointerExceptionWithMessage() {

thrown.expect(NullPointerException.class);

thrown.expectMessage("happened?");

thrown.expectMessage(startsWith("What"));

throw new NullPointerException("What happened?");

}

}

**ClassRule**

La anotación ClassRule amplía la idea de Reglas a nivel de método, agregando campos estáticos que pueden afectar el funcionamiento de una clase completa. Cualquier subclase de ParentRunner, incluidas las clases estándar BlockJUnit4ClassRunner y Suite, admitirá ClassRules.

Por ejemplo, aquí hay un conjunto de pruebas que se conecta a un servidor una vez antes de que se ejecuten todas las clases de prueba y se desconecta una vez que finalizan:

@RunWith(Suite.class)

@SuiteClasses({A.class, B.class, C.class})

public class UsesExternalResource {

public static final Server myServer = new Server();

@ClassRule

public static final ExternalResource resource = new ExternalResource() {

@Override

protected void before() throws Throwable {

myServer.connect();

};

@Override

protected void after() {

myServer.disconnect();

};

};

}

**RuleChain**

La regla de la cadena de reglas permite ordenar las reglas de prueba:

public static class UseRuleChain {

@Rule

public final TestRule chain = RuleChain

.outerRule(new LoggingRule("outer rule"))

.around(new LoggingRule("middle rule"))

.around(new LoggingRule("inner rule"));

@Test

public void example() {

assertTrue(true);

}

}

**Custom Rules**

Por supuesto, el poder de implementar TestRule proviene de usar una combinación de constructores personalizados, agregar métodos a la clase para usar en las pruebas y envolver la declaración proporcionada en una nueva declaración. Por ejemplo, considere la siguiente regla de prueba que proporciona un registrador con nombre para cada prueba:

public class TestLogger implements TestRule {

private Logger logger;

public Logger getLogger() {

return this.logger;

}

@Override

public Statement apply(final Statement base, final Description description) {

return new Statement() {

@Override

public void evaluate() throws Throwable {

logger = Logger.getLogger(description.getTestClass().getName() + '.' + description.getDisplayName());

base.evaluate();

}

};

}

}

public class MyLoggerTest {

@Rule

public final TestLogger logger = new TestLogger();

@Test

public void checkOutMyLogger() {

final Logger log = logger.getLogger();

log.warn("Your test is showing!");

}

}