JUnit 5 - Assert Class

Br. Rafael Rodríguez Guzmán

Assert Class

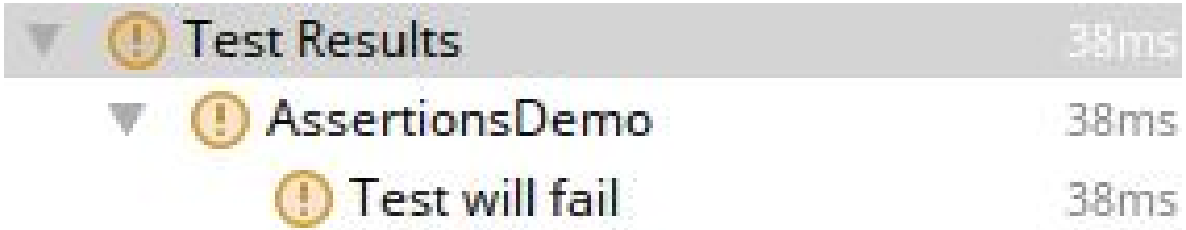
JUnit 5 brinda métodos estáticos para probar ciertas condiciones usando la clase *Assert*. Estas sentencias comienzan con *assert*, y permiten especificar el mensaje de error y el resultado esperado y actual.

Un método *assert* compara el valor retornado por la prueba contra el valor esperado. Lanza una excepción *AssertionException* si la comparación falla.

`fail(String message)`

Permite a la prueba fallar. Puede ser usado para revisar que alguna parte específica del código no está siendo alcanzada, o para ejecutar una prueba fallida antes de que el código sea implementado.

```java
@Test
@DisplayName("Test will fail")
void failTestDemo() {
   fail("Failed test");
}
```



`assertTrue(boolean condition, String message)`

Revisa si la condición dada es verdadera. Mensaje opcional.

```
@Test
@DisplayName("Assert true test")
void assertTrueDemo() {
   String nombre = "walook";

   assertTrue(nombre.startsWith("w"));
}
```

`assertFalse(boolean condition, String message)`

Revisa si la condición dada es falsa. Mensaje opcional.

```
@Test
@DisplayName("Assert false test")
void assertFalseDemo() {
   String nombre = "walook";

   assertFalse(nombre.startsWith("r"));
}
```

`assertEquals(Object expected, Objectactual, String message)`

Prueba que el valor esperado y actual sean equivalentes. El mensaje es opcional.

```
@Test
@DisplayName("Assert equals test")
void assertEqualsDemo() {
   int valor1 = 5, valor2 = 10, resultado = 15;

   assertEquals(resultado, valor1 + valor2, "Resultado: " + resultado);
}
```

`assertNull(Object, String message)`

Valida que el objeto sea nulo. Mensaje opcional.

```
@Test
@DisplayName("Assert null test")
void assertNullDemo() {
   Person person = null;

   assertNull(person);
}
```

`assertNotNull(Object, String message)`

Valida que el objeto no sea nulo. Mensaje opcional.

```
@Test
@DisplayName("Assert not null test")
void assertNotNullDemo() {
   Person person = new Person();

   assertNotNull(person);
}
```

`assertSame(Object expected, Object actual, String message)`

Valida que dos variables hacen referencia al mismo objeto. Mensaje opcional.

```
@Test
@DisplayName("Assert same test")
void assertSameDemo() {
   Person original = new Person();
   Person prepoceso = original;
   Person postproceso = original;

   assertSame(prepoceso, postproceso);
}
```

`assertNotSame(Object expected, Object actual, String message)`

Valida que dos variables no hacen referencia al mismo objeto. Mensaje opcional.

```
@Test
@DisplayName("Assert not same test")
void assertNotSameDemo() {
   Person prepoceso = new Person();
   Person postproceso = new Person();

   assertNotSame(prepoceso, postproceso);
}
```

`assertAll(Executable...executables)`

Valida que todos los ejecutables dados no lancen excepciones.

```
@Test
@DisplayName("Assert all test")
void assertAllDemo() {
   Person person = new Person("John", "Doe");

   assertAll("person",
         () -> assertNotNull(person),
         () -> assertEquals("John", person.getFirstName())
   );
}
```

```
assertArrayEquals(Object[] expected, Object[] actual,
[double delta], [String message])
```

Valida que el arreglo esperado sea igual al arreglo actual. Los arreglos pueden ser de tipos primitivos (int, double, float, char, boolean, byte, long, short).

En el caso de de comparar arreglos de tipo double, es necesario usar double delta, que especifica la precisión de decimales con la que se debe comparar.

El mensaje es opcional.

```
@Test
@DisplayName("Assert array equals test")
void assertArrayEqualsDemo() {
   char[] array1= {'H', 'E', 'L', 'L', 'O'};
   char[] array2= {'H', 'E', 'L', 'L', 'O'};

   assertArrayEquals(array1, array2, "Arreglos iguales.");
}
```

`assertThrows(Class<T> expectedType, Executable executable, String message)`

Verifica que la ejecución del ejecutable proporcionado lance una excepción del tipo especificado. Mensaje opcional.

```
@Test
@DisplayName("Assert throw excepcion test")
void assertThrowsException() {
   Throwable exception = assertThrows(IllegalArgumentException.class, () -> {
      throw new IllegalArgumentException("a message");
   });
   assertEquals("a message", exception.getMessage());
}
```

```
assertTimeout(Duration timeout, Executable executable,
String message)
```

Valida que la ejecución del ejecutable proporcionado termine antes del tiempo especificado. Mensaje opcional.

El ejecutable se ejecutará en el mismo hilo que el código de llamada. En consecuencia, la ejecución del ejecutable no se abortará de manera preventiva si se excede el tiempo de espera.

```java
@Test
@DisplayName("Assert timeout test")
void assertTimeoutDemo() {
   assertTimeout(ofMinutes(2), () -> {
      //Tarea a ejecutar de duración menor a 2 minutos.
      Person person = new Person();
      System.out.println(person.getFirstName() + " " + person.getLastName());
   });
}
```

```
assertTimeoutPreemtively(Duration timeout, Executable
executable, String message)
```

Valida que la ejecución del ejecutable proporcionado termine antes del tiempo especificado. Mensaje opcional.

El ejecutable se ejecutará en un hilo diferente al del código de llamada. Además, la ejecución del ejecutable será anulada preventivamente si se excede el tiempo de espera.

```java
@Test
@DisplayName("Assert tiemeout preemtively test")
void assertTimeoutPreemtivelyDemo() {
    assertTimeoutPreemptively(ofMillis(10), () -> {
        // Simula una tarea que dura más de 10 milisegundos.
        Thread.sleep(100);
    });
}
```

Referencias

Class Assertions. Recuperado de: http://junit.org/junit5/docs/current/api/org/junit/jupiter/api/Assertions.html