

Prueba con errores ya que `tester.getFrequency` devuelve true si es FM y false si es AM, no devuelve un String. Además, `getStation` devuelve un double por lo que debe existir tolerancia en las pruebas.

```
1 package application;
2 import static org.junit.jupiter.api.Assertions.assertEquals;
5
6 public class MyTests {
7
8     @Test
9     public void radioTester() {
10         Radio tester = new Radioimp(); // MyClass is tested
11
12         // assert statements
13         //Aquí se ejecutara nuestro test.
14         tester.toggle();
15         assertEquals(true, tester.getState(), "Se acaba de prender el radio. Debe estar encendido");
16         tester.changeStation(true);
17         tester.changeStation(true);
18         tester.saveStation(3);
19         tester.changeFrequency();
20         tester.changeStationButton(3);
21         tester.getStation();
22         assertEquals(true, tester.getFrequency(), "Debe estar en esta frecuencia FM.");
23         assertEquals(88.3, tester.getFrequency(), "Debe estar en esta estacion que se guarda en el 3.");
24     }
25 }
```

Failure Trace

AssertionFailedError: Debe estar en esta estacion que se guarda en el 3. ==> expected: <FM> but was: <false>

Tests: radioTester(MyTests.java:22)

.ForEachOps\$ForEachOp\$OpRef.accept(Unknown Source)

ReferencePipeline\$2\$1.accept(Unknown Source)

.forEachRemaining(Unknown Source)

Iterators\$Iterator\$Splitter.forEachRemaining(Unknown Source)

AbstractPipeline.copyInto(Unknown Source)

AbstractPipeline.wrapAndCopyInto(Unknown Source)

.ForEachOps\$ForEachOp\$OpRef.evaluateSequential(Unknown Source)

.ForEachOps\$ForEachOp\$OpRef.evaluateSequential(Unknown Source)

AbstractPipeline.evaluate(Unknown Source)

```
1 package application;
2 import static org.junit.jupiter.api.Assertions.assertEquals;
5
6 public class MyTests {
7
8     @Test
9     public void radioTester() {
10         Radio tester = new Radioimp(); // MyClass is tested
11
12         // assert statements
13         //Aquí se ejecutara nuestro test.
14         tester.toggle();
15         assertEquals(true, tester.getState(), "Se acaba de prender el radio. Debe estar encendido");
16         tester.changeStation(true);
17         tester.changeStation(true);
18         tester.saveStation(3);
19         tester.changeFrequency();
20         tester.changeStationButton(3);
21         tester.getStation();
22         assertEquals(true, tester.getFrequency(), "Debe estar en esta frecuencia FM.");
23         assertEquals(88.3, tester.getStation(), "a estacion que se guarda en el 3.");
24     }
25 }
```

Failure Trace

AssertionFailedError: a estacion que se guarda en el 3. ==> expected: <88.3> but was: <88.30000000000001>

Tests: radioTester(MyTests.java:23)

.ForEachOps\$ForEachOp\$OpRef.accept(Unknown Source)

ReferencePipeline\$2\$1.accept(Unknown Source)

.forEachRemaining(Unknown Source)

Iterators\$Iterator\$Splitter.forEachRemaining(Unknown Source)

AbstractPipeline.copyInto(Unknown Source)

AbstractPipeline.wrapAndCopyInto(Unknown Source)

.ForEachOps\$ForEachOp\$OpRef.evaluateSequential(Unknown Source)

.ForEachOps\$ForEachOp\$OpRef.evaluateSequential(Unknown Source)

AbstractPipeline.evaluate(Unknown Source)

Prueba finalmente ejecuta sin errores de nuestro radio.

