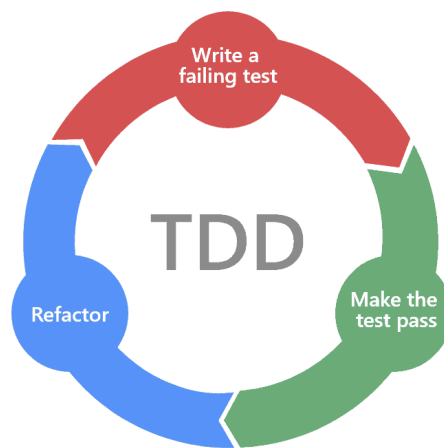


Práctica 2 - Test Driven Development



Ampliación de Ingeniería del Software

Curso 2020-2021

Fátima Ezahra Smounat Mahidar
Milagros Mouriño Ursul

Índice

1. Primer ciclo	1
2. Segundo ciclo	4
3. Tercer ciclo	7
4. Cuarto ciclo	9
5. Quinto ciclo	12
6. Sexto ciclo	15
7. Séptimo ciclo	16
8. Octavo ciclo	17
9. Noveno ciclo	19
10. Décimo ciclo	20
11. Onceavo ciclo	23
12. Doceavo ciclo	24
13. Treceavo ciclo	25
14. Catorceavo ciclo	27
15. Arreglos	28

1. Primer ciclo

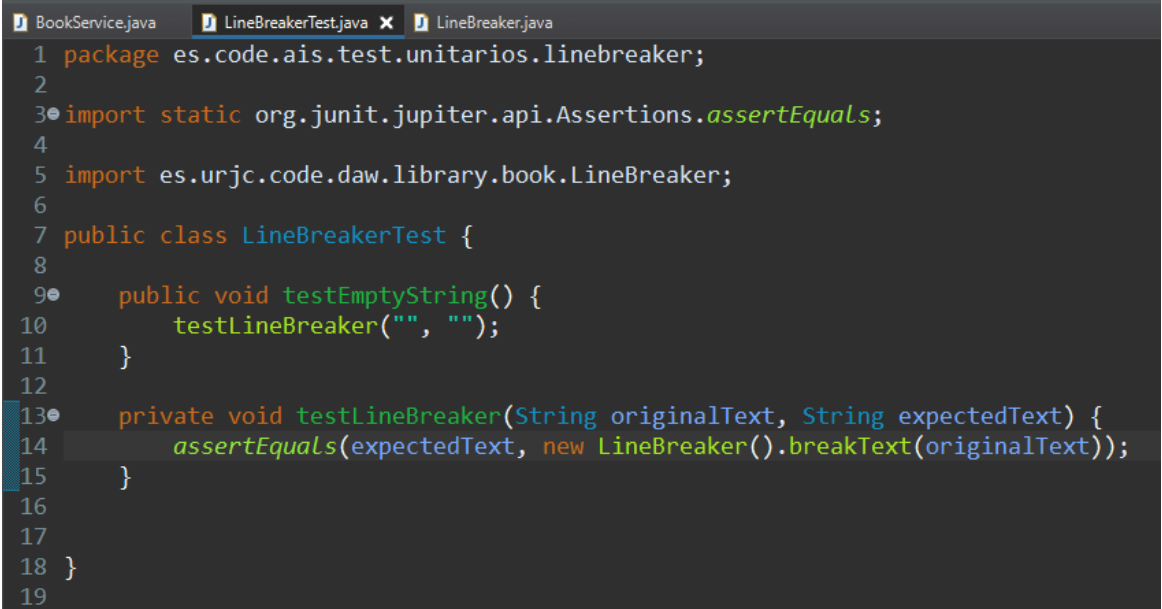
Lo primero que hemos hecho ha sido crear la clase **LineBreakerTest** dentro del paquete **es.code.ais.test.unitarios.linebreaker** en **src/test/java**.

Dado que en la práctica se nos proporciona el listado de los catorce ejemplos a testear hemos proseguido con el segundo paso: implementar el primer test de la cadena vacía.

Se dedujo que se necesitaría un test genérico que hemos denominado **testLineBreaker**, que comprobaría la funcionalidad a implementar. Este recibe como parámetros el texto original (**originalText**) y el resultado esperado (**expectedText**). El método **assertEquals** verifica si el resultado esperado que se pasó como parámetro se corresponde al que devuelve la llamada del método que implementamos.

Al escribir el código, instantáneamente se daban errores de compilación. El primero de ellos era que no se reconocía la clase **LineBreaker**, dado que ésta no estaba creada. Automáticamente la generamos dentro del paquete **es.urjc.code.daw.library.book** en **src/main/java** y aplicamos lo mismo con el método **breakText**.

De manera adicional, tuvimos que importar el **assertEquals**.



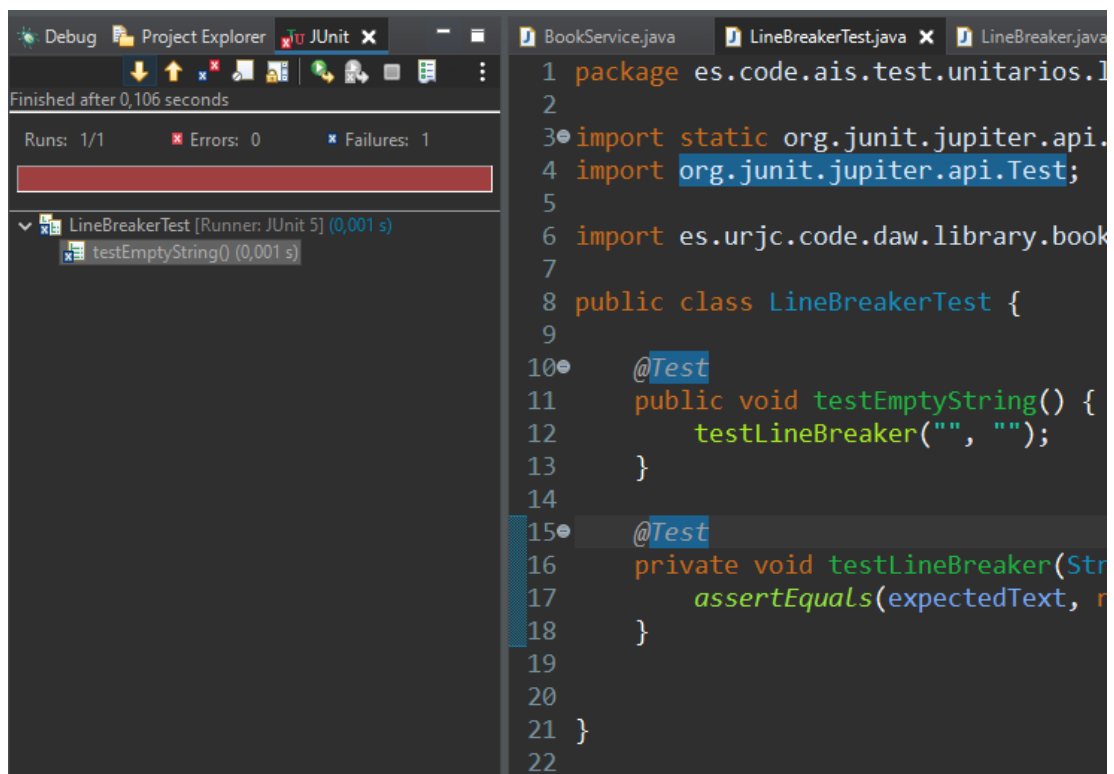
```
1 package es.code.ais.test.unitarios.linebreaker;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
5 import es.urjc.code.daw.library.book.LineBreaker;
6
7 public class LineBreakerTest {
8
9     public void testEmptyString() {
10         testLineBreaker("", "");
11     }
12
13     private void testLineBreaker(String originalText, String expectedText) {
14         assertEquals(expectedText, new LineBreaker().breakText(originalText));
15     }
16
17
18 }
19
```

```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public Object breakText(String originalText) {
6         // TODO Auto-generated method stub
7         return null;
8     }
9
10 }
11

```

Intentamos ejecutar el test y nos damos cuenta de que nos faltan las anotaciones `@Test`. Tras haberlas colocado sobre las cabeceras de los métodos, volvemos a ejecutar la clase y vemos que falla el único test creado hasta el momento, ya que el método está sin implementar.



Al visualizar de nuevo el método, nos dimos cuenta de que no habíamos incluido el número de líneas en los que se divide la descripción. Además, cambiamos el tipo que devuelve el método (`String`) y renombramos el primer parámetro a `text`, tal y como se especifica en el enunciado:

```
BookService.java *LineBreakerTest.java LineBreaker.java UnitaryTest.java
1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         return null;
7     }
8
9 }
10
```

Implementamos el código mínimo e indispensable para que el test se ponga en verde haciendo que el método devuelva "" en lugar de null. Observamos que ahora pasan los tests:

```
*LineBreaker.java
1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         return "";
7     }
8
9 }
10
```

```
Finished after 0,067 seconds
Runs: 1/1 Errors: 0 Failures: 0
LineBreakerTest [Runner: JUnit 5] (0,015 s)
  testEmptyString() (0,015 s)

1 package es.code.ais.test.unitarios.linebre
2
3 import static org.junit.jupiter.api.Assert
4 import org.junit.jupiter.api.Test;
5
6 import es.urjc.code.daw.library.book.Line
7
8 public class LineBreakerTest {
9
10     @Test
11     public void testEmptyString() {
12         testLineBreaker("", "");
13     }
14
15     @Test
16     private void testLineBreaker(String o
17         assertEquals(expectedText, new Li
18     }
```

Pasamos a refactorizar el código:

```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         if (text.equals("")) {
7             return "";
8         }
9         return null;
10    }
11
12 }
13

```

Dado que, independientemente del número de líneas, si el código es la cadena vacía entonces se devuelve esta misma. Utilizamos **return null** para que no den errores de compilación y podamos seguir aplicando TDD.

2. Segundo ciclo

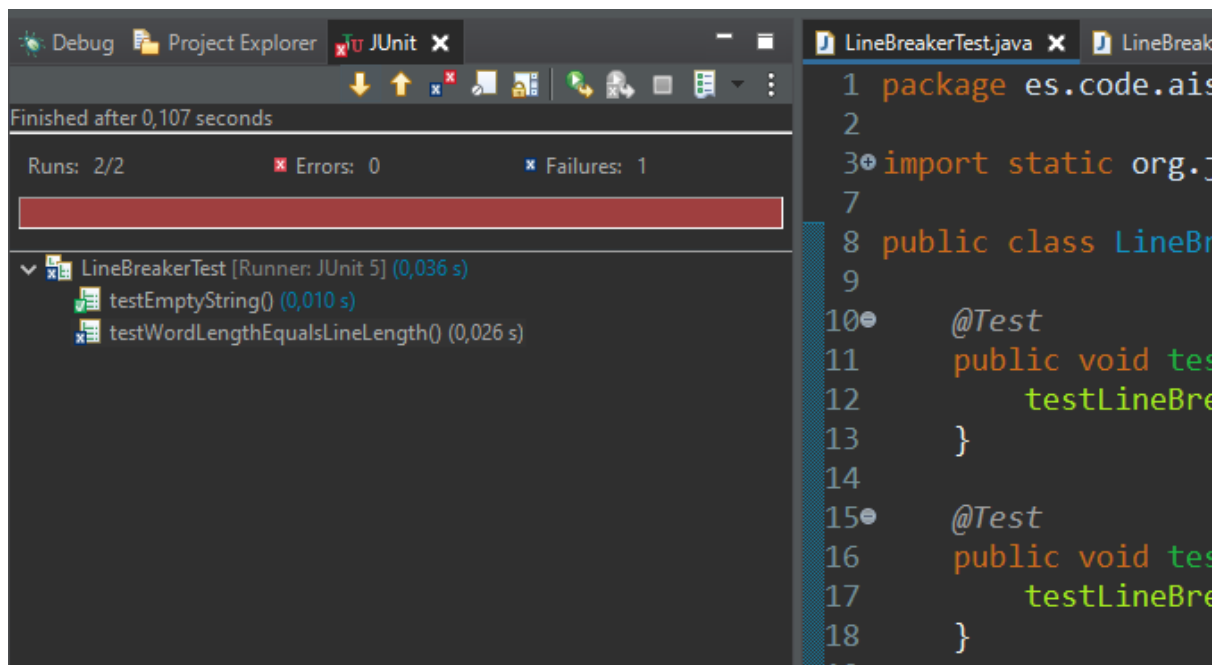
Creamos el test correspondiente al segundo ejemplo denominado **testWordLengthEqualsLineLength**.

```

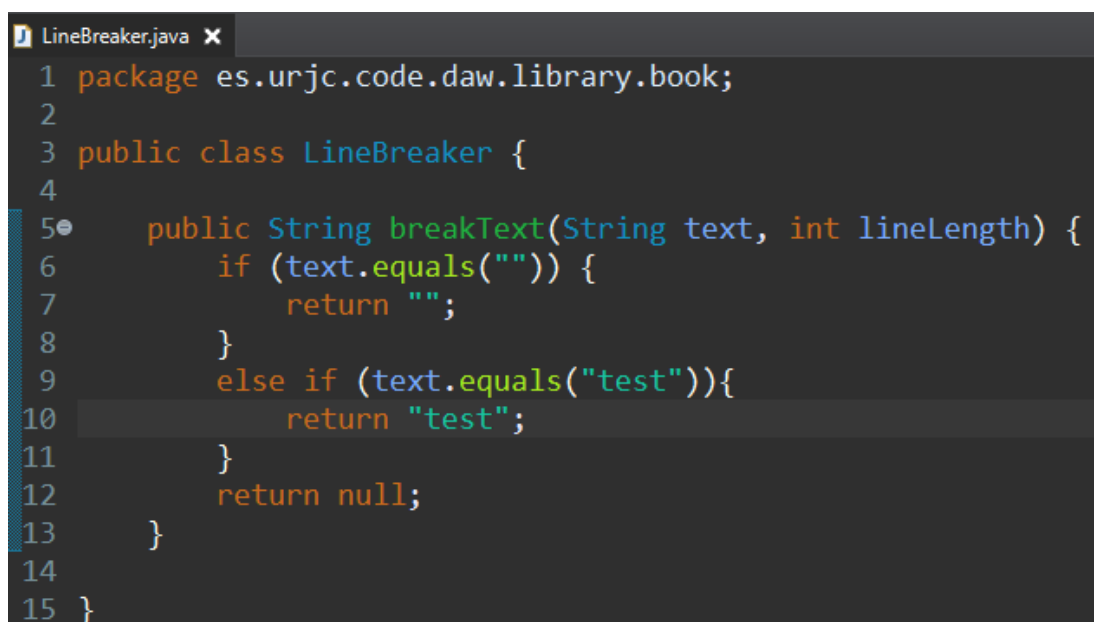
1 package es.code.ais.test.unitarios.linebreaker;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
5 public class LineBreakerTest {
6
7     @Test
8     public void testEmptyString() {
9         testLineBreaker("", "", 2);
10    }
11
12     @Test
13     public void testWordLengthEqualsLineLength() {
14         testLineBreaker("test", "test", 4);
15    }
16
17     @Test
18     private void testLineBreaker(String originalText, String expectedText, int lineLength) {
19         assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength));
20    }
21
22 }
23

```

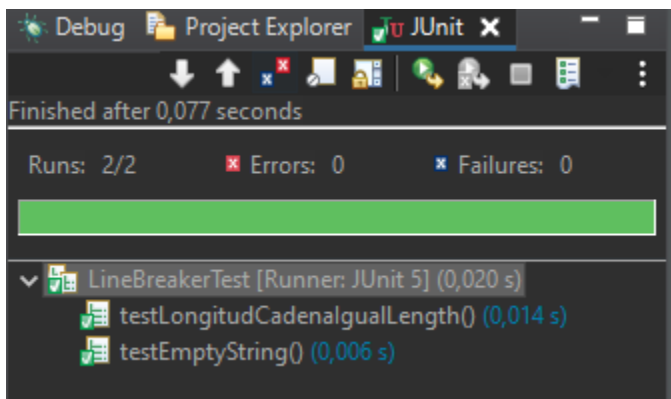
Vemos que el test falla:



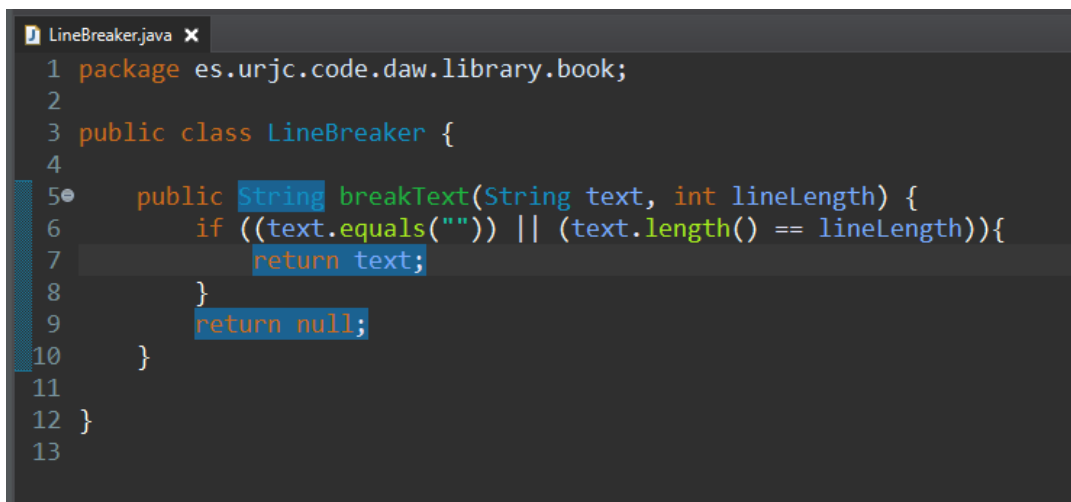
Escribimos el código mínimo y necesario para que el test pase:



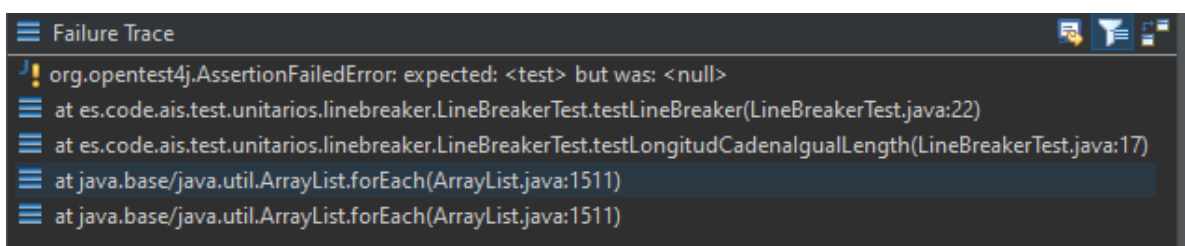
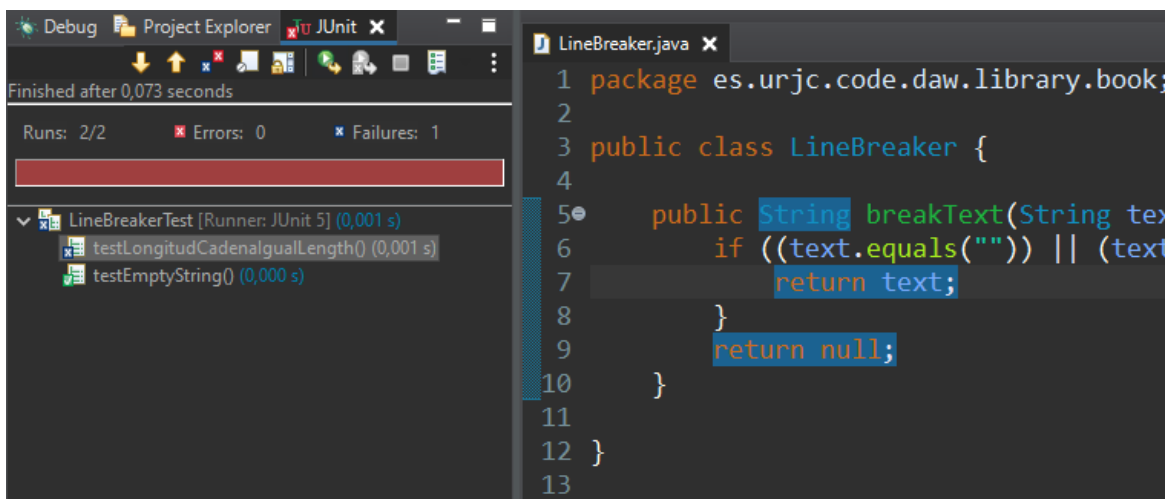
Observamos que pasan los dos tests:



Refactorizamos el código:



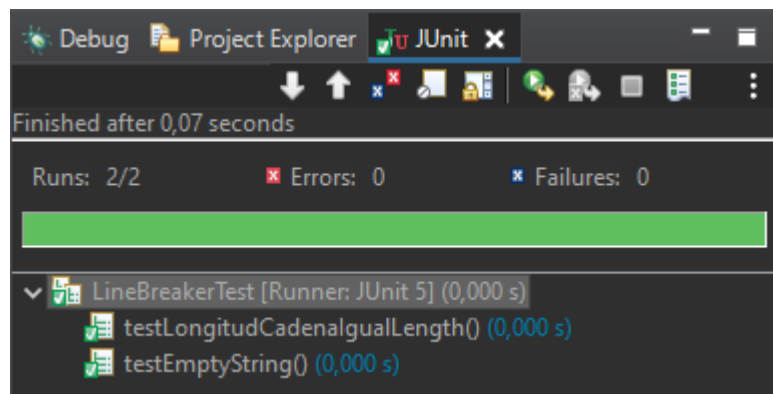
Obtenemos que **testWordLengthEqualsLineLength** falla:



Luego entendimos que esto se debía a que el parámetro de **lineLength** lo pusimos igual a 10 para todos los casos cuando teníamos que colocar un 2 y 4 para los tests de los casos implementados. Ante esto, hemos decidido añadir el parámetro **lineLength** en **testLineBreaker**:

```
LineBreakerTest.java x
1 package es.code.ais.test.unitarios.linebreaker;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
5
6
7
8 public class LineBreakerTest {
9
10     @Test
11     public void testEmptyString() {
12         testLineBreaker("", "", 2);
13     }
14
15     @Test
16     public void testLongitudCadenaIgualLength() {
17         testLineBreaker("test", "test", 4);
18     }
19
20     @Test
21     private void testLineBreaker(String originalText, String expectedText, int lineLength) {
22         assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength));
23     }
24 }
```

Observamos que ya sí pasan ambos tests:



3. Tercer ciclo

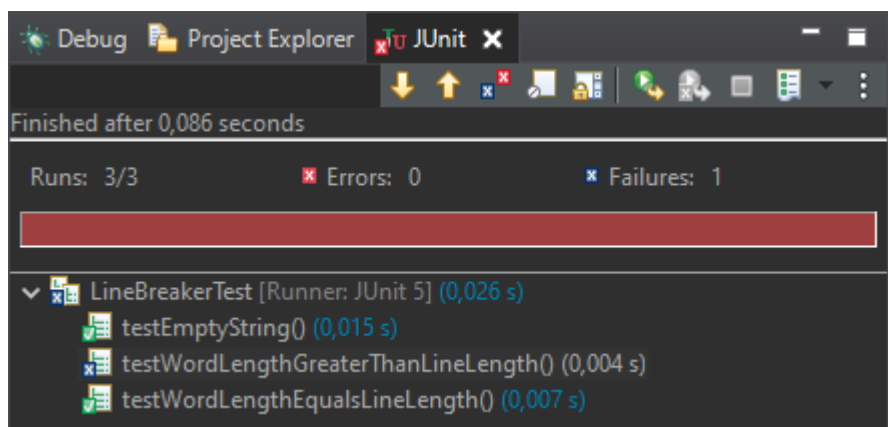
Primero creamos el test correspondiente al tercer ejemplo:

```

30 import static org.junit.jupiter.api.Assertions.assertEquals;
7
8 public class LineBreakerTest {
9
10     @Test
11     public void testEmptyString() {
12         testLineBreaker("", "", 2);
13     }
14
15     @Test
16     public void testWordLengthEqualsLineLength() {
17         testLineBreaker("test", "test", 4);
18     }
19
20     @Test
21     public void testWordLengthGreaterThanLineLength() {
22         testLineBreaker("test", "test", 5);
23     }
24
25     @Test
26     private void testLineBreaker(String originalText, String expectedText, int lineLength) {
27         assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength));
28     }
29

```

Al ejecutar los tests, observamos que **testWordLengthGreaterThanLineLength** falla:

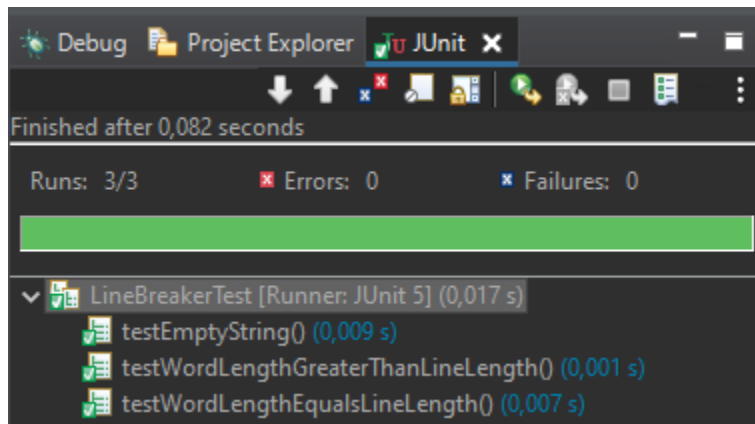


Implementamos lo mínimo para hacer que el test no falle:

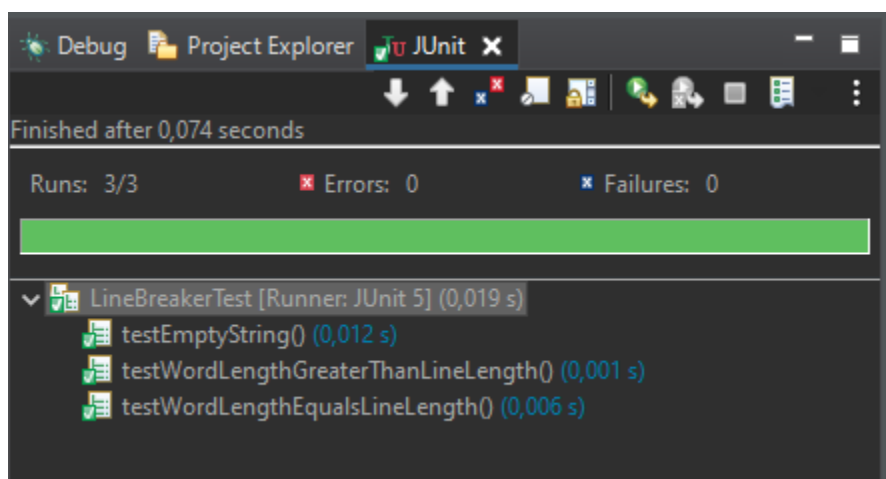
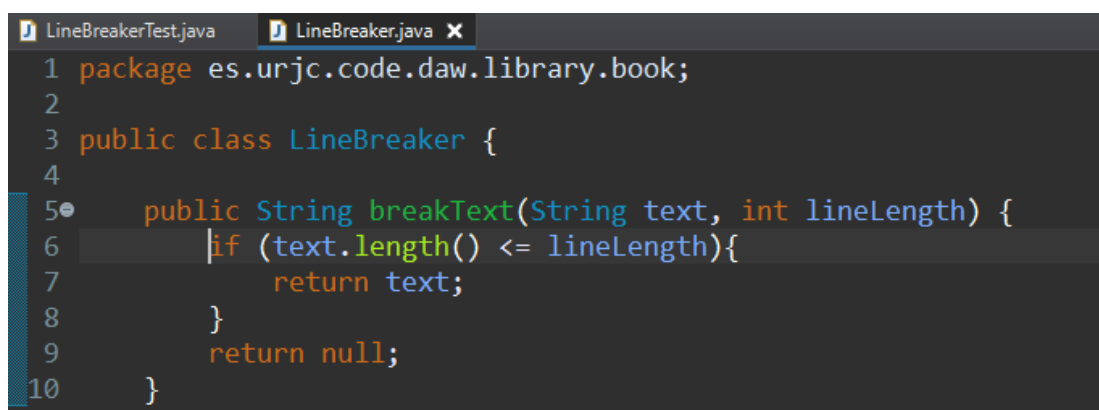
```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         if ((text.equals("")) || (text.length() == lineLength)){
7             return text;
8         }
9         else if (text.length() < lineLength) {
10             return text;
11         }
12         return null;
13     }
14

```



Refactorizamos el código:



4. Cuarto ciclo

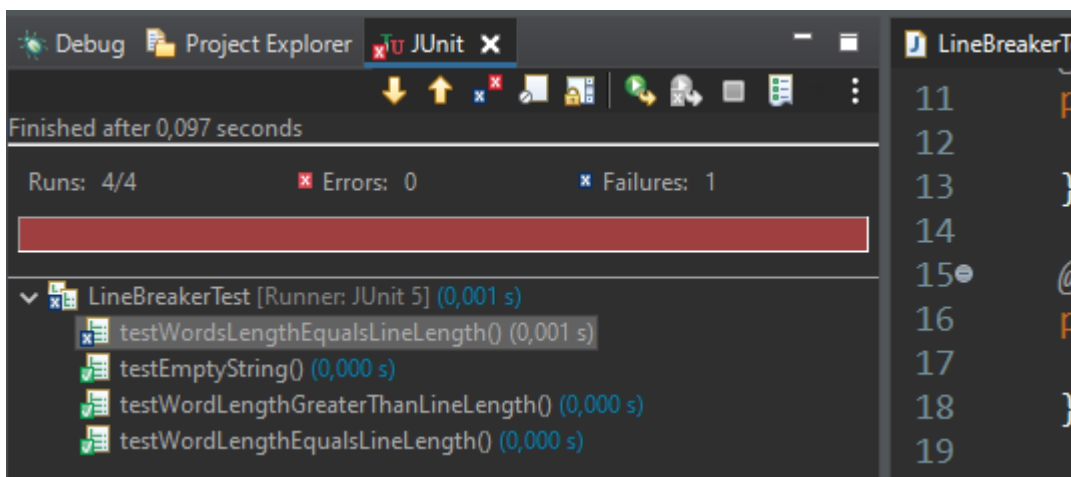
Elaboramos el test denominado **testWordsLengthEqualsLineLength** correspondiente al cuarto ejemplo:

```

15• @Test
16 public void testWordLengthEqualsLineLength() {
17     testLineBreaker("test", "test", 4);
18 }
19
20• @Test
21 public void testWordLengthGreaterThanLineLength() {
22     testLineBreaker("test", "test", 5);
23 }
24
25• @Test
26 public void testWordsLengthEqualsLineLength() {
27     testLineBreaker("test test", "test\ntest", 4);
28 }
29
30• @Test
31 private void testLineBreaker(String originalText, String expectedText, int lineLength) {
32     assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength));
33 }
34

```

Comprobamos que el test falla:

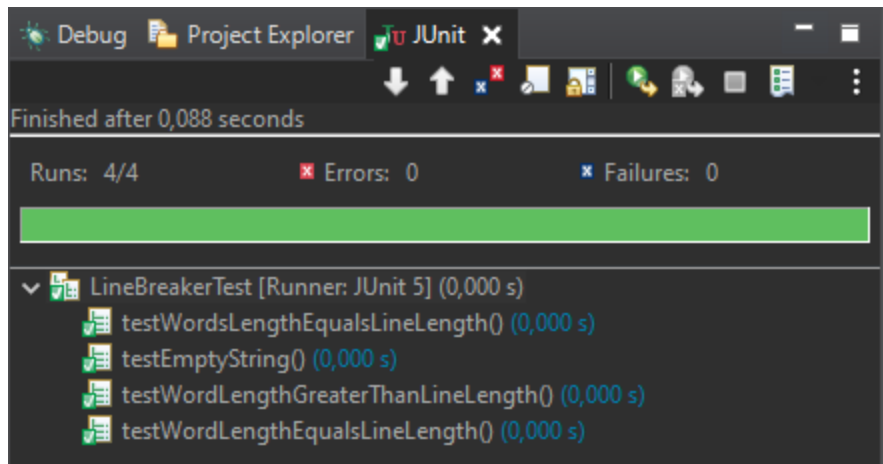


Intentamos que el test pase:

```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         if (text.length() <= lineLength){
7             return text;
8         }
9         else if (text.equals("test test")) {
10             return "test\ntest";
11         }
12         return null;
13     }
14 }

```



Al refactorizar, observamos que cuando el carácter que se encuentra en la posición de **lineLength** es el **espacio en blanco** nos encontramos ante un **caso recursivo**. Para poder implementar el código relativo a este, lo primero que hicimos fue ponernos ejemplos donde se cumpliese esa condición:

```
breakText(test test, 4) = test\ntest
```

```
breakText(test test test, 4) = test\ntest\ntest
```

```
breakText(test test test test, 9) = test test\ntest test
```

```
breakText(test test test test test test test test test, 19) = test test test test\ntest test test test
```

```
breakText(test test test test test test test test, 19) = test test test test\ntest test test
```

```
breakText(test test test, 9) = test test\ntest
```

```
breakText(test test test test, 4) = test\ntest\ntest\ntest
```

Observamos los patrones:

```
breakText(test test, 4) = test\ntest = breakText(test, 4) + '\n' + breakText(test, 4)
```

```
breakText(test test test, 4) = test\ntest\ntest = breakText(test, 4) + '\n' + breakText(test, 4) + '\n' + breakText(test, 4) = breakText(test, 4) + '\n' + breakText(test test, 4)
```

```
breakText(test test test test, 9) = test test\ntest test = breakText(test test, 9) + '\n' + breakText(test test, 9)
```

```
breakText(test test test test test test test test test, 19) = test test test test\ntest test test test =
```

```
breakText(test test test test, 19) + '\n' + breakText(test test test test, 19)
```

```
breakText(test test test test test test test test, 19) = test test test test\ntest test test =
```

```
breakText(test test test test, 19) + '\n' + breakText(test test test, 19)
```

```
breakText(test test test, 9) = test test\ntest = breakText(test test, 9) + '\n' + breakText(test, 9)
```

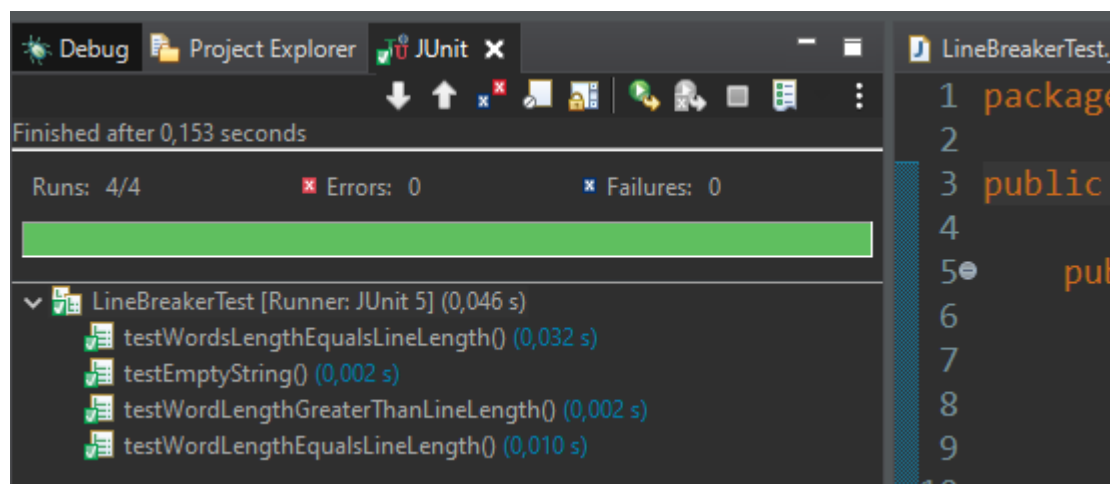
`breakText(test test test test, 4) = test\ntest\ntest\ntest = breakText(test, 4) + '\n' + breakText(test, 4) + '\n' + breakText(test, 4) + '\n' + breakText(test, 4) = breakText(test, 4) + '\n' + breakText(test test test, 4)`

Concluimos que la resolución del problema sería la **concatenación** de la llamada al **caso base** `breakText(text.substring(0, lineLength), lineLength)`, el **salto de línea** y el **caso recursivo**.

Por último, cabe mencionar que nos hemos dado cuenta de que el **caso siete** se resuelve de la misma forma, ya que en `lineLength` nos encontramos con un **blanco**.

```
1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         if (text.length() <= lineLength){
7             return text;
8         }
9         else if (text.charAt(lineLength) == ' ') {
10             return breakText(text.substring(0, lineLength), lineLength) + '\n'+
11                    breakText(text.substring(lineLength+1, text.length()), lineLength);
12         }
13         else{
14             return null;
15         }
16     }
17 }
```

Comprobamos que las cuatro pruebas pasan:



5. Quinto ciclo

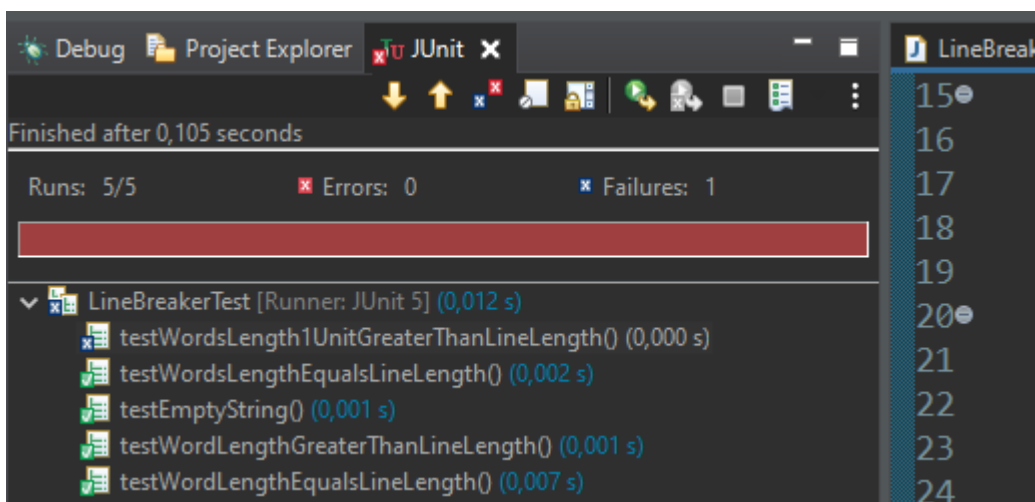
Creamos el test para el quinto ejemplo:

```

20●  @Test
21  public void testWordLengthGreaterThanOrEqualToLineLength() {
22      testLineBreaker("test", "test", 5);
23  }
24
25●  @Test
26  public void testWordsLengthEqualsLineLength() {
27      testLineBreaker("test test", "test\ntest", 4);
28  }
29
30●  @Test
31  public void testWordsLength1UnitGreaterThanOrEqualToLineLength() {
32      testLineBreaker("test test", "test\ntest", 5);
33  }
34
35●  @Test
36  private void testLineBreaker(String originalText, String expectedText, int lineLength) {
37      assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength));
38  }
39

```

Observamos que el test falla:

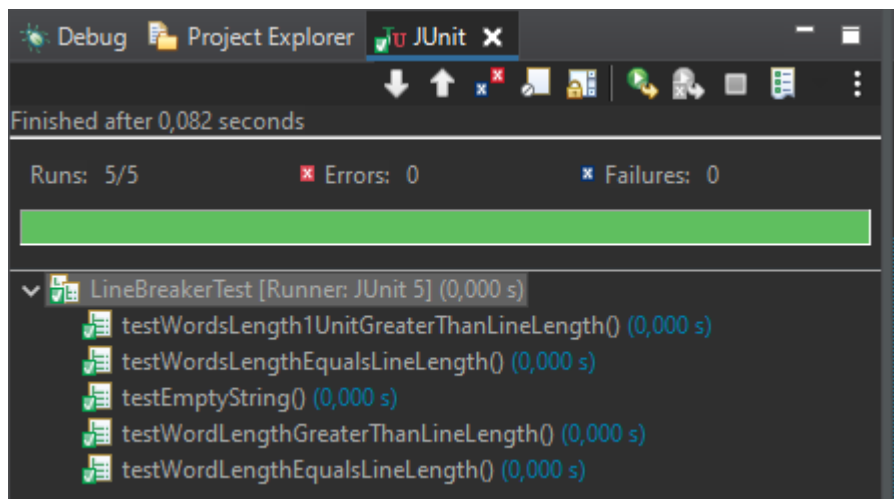


Procuramos que el test pase:

```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         if (text.length() <= lineLength){
7             return text;
8         }
9         else if (text.charAt(lineLength) == ' ') {
10             return breakText(text.substring(0, lineLength), lineLength) + '\n'+
11                 breakText(text.substring(lineLength+1, text.length()), lineLength);
12         }
13         else{
14             return "test\ntest";
15         }
16     }
17

```



Refactorizamos y nos percatamos de que el código escrito en el **ciclo 4** se puede hacer más eficiente. Nos podemos ahorrar una llamada recursiva a la pila sustituyendo **breakText(text.substring(0, lineLength))** por **text.substring(0, lineLength)**.

```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         if (text.length() <= lineLength){
7             return text;
8         }
9         else if (text.charAt(lineLength) == ' ') {
10             return text.substring(0, lineLength) + '\n'+
11                 breakText(text.substring(lineLength+1, text.length()), lineLength);
12         }
13         else{
14             return breakText(text, lineLength-1);
15         }
16     }
17 }

```

Con respecto a este ciclo, observamos el patrón que se sigue para algunos ejemplos:

`breakText(test test, 5) = breakText(test test, 4) = test\ntest`

`breakText(test test, 6) = breakText(test test, 5) = breakText(test test, 4) = test\ntest`

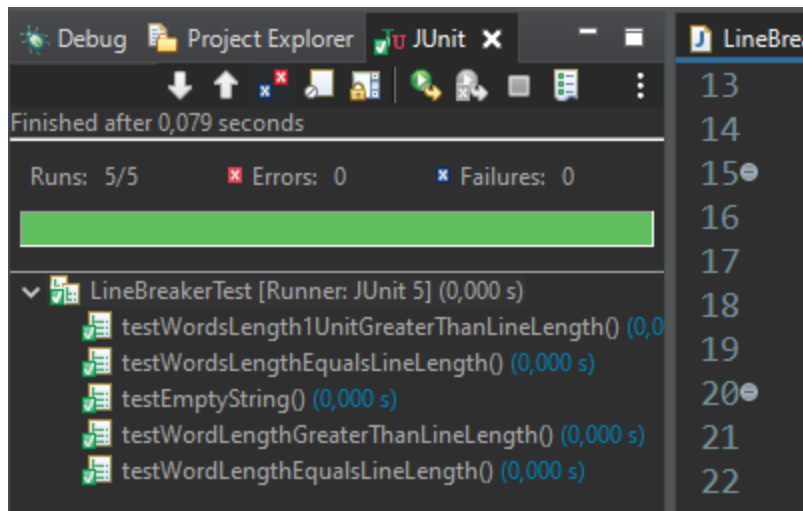
`breakText(fatima y milagros, 10)= fatima y\nmilagros = breakText(fatima y milagros, 9) = breakText(fatima y milagros, 8) = fatima y\nmilagros`

`breakText(test test test test, 6) = breakText(test test test test, 5) =`

`breakText(test test test test, 4) = test\ntest\ntest\ntest`

Nos damos cuenta aquí también por el segundo ejemplo analizado que se corresponde con el ejemplo del sexto ciclo, luego los tests van a pasar también con el código implementado.

Observamos que pasan los tests:



6. Sexto ciclo

Escribimos el código para el test del sexto ejemplo:

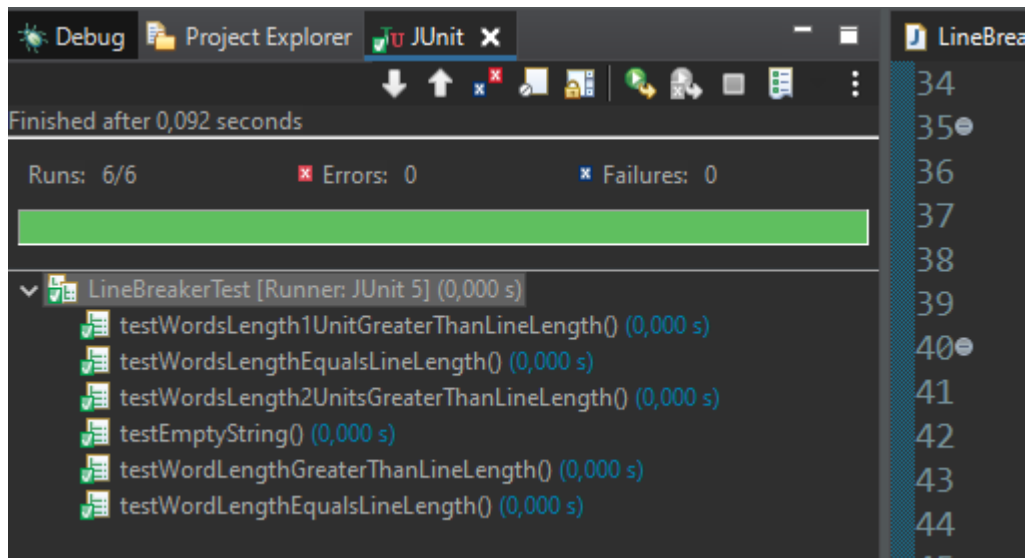
```
@Test
public void testWordsLengthEqualsLineLength() {
    testLineBreaker("test test", "test\ntest", 4);
}

@Test
public void testWordsLength1UnitGreaterThanLineLength() {
    testLineBreaker("test test", "test\ntest", 5);
}

@Test
public void testWordsLength2UnitsGreaterThanLineLength() {
    testLineBreaker("test test", "test\ntest", 6);
}

@Test
private void testLineBreaker(String originalText, String expectedText, int lineLength) {
    assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength));
}
```

Al ejecutarlo, pasan todos los tests porque el ejemplo anterior es análogo a este, luego el código escrito hace que no sea necesario implementar nuevo código:



7. Séptimo ciclo

Realizamos el test correspondiente al séptimo ciclo:

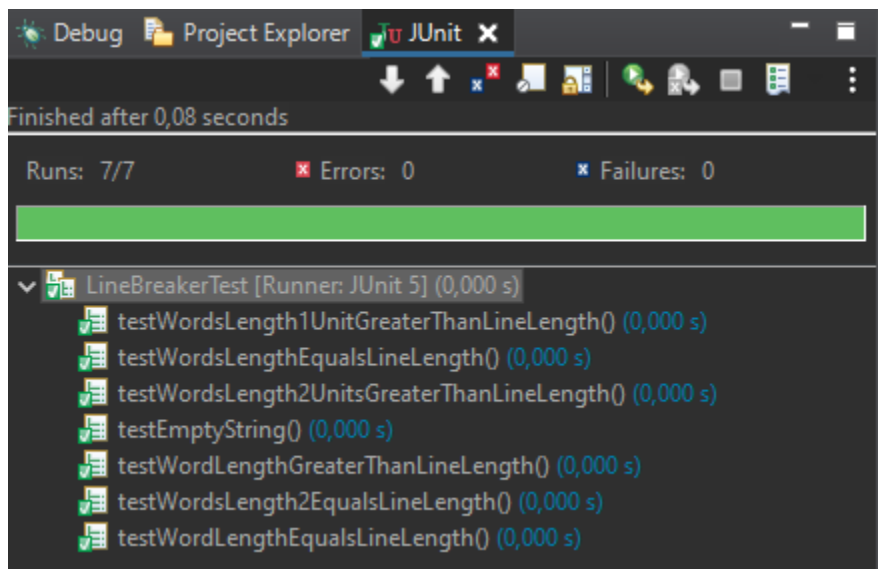
```
@Test
public void testWordsLength1UnitGreaterThanLineLength() {
    testLineBreaker("test test", "test\ntest", 5);
}

@Test
public void testWordsLength2UnitsGreaterThanLineLength() {
    testLineBreaker("test test", "test\ntest", 6);
}

@Test
public void testWordsLength2EqualsLineLength() {
    testLineBreaker("test test test test", "test test\ntest test", 9);
}

@Test
private void testLineBreaker(String originalText, String expectedText, int lineLength) {
    assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength));
}
```

Al igual que en el caso anterior, los tests pasan porque el caso 4 se resuelve igual a este:



8. Octavo ciclo

Implementamos el test correspondiente al octavo ciclo:

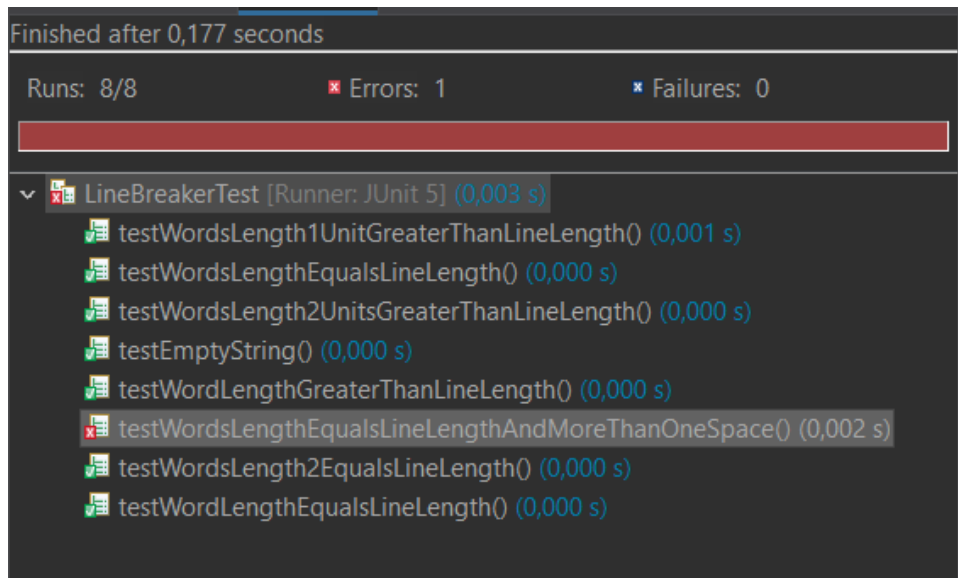
```
@Test
public void testWordsLength2UnitsGreaterThanLineLength() {
    testLineBreaker("test test", "test\ntest", 6);
}

@Test
public void testWordsLength2EqualsLineLength() {
    testLineBreaker("test test test test", "test test\ntest test", 9);
}

@Test
public void testWordsLengthEqualsLineLengthAndMoreThanOneSpace() {
    testLineBreaker("test  test", "test\ntest", 4);
}

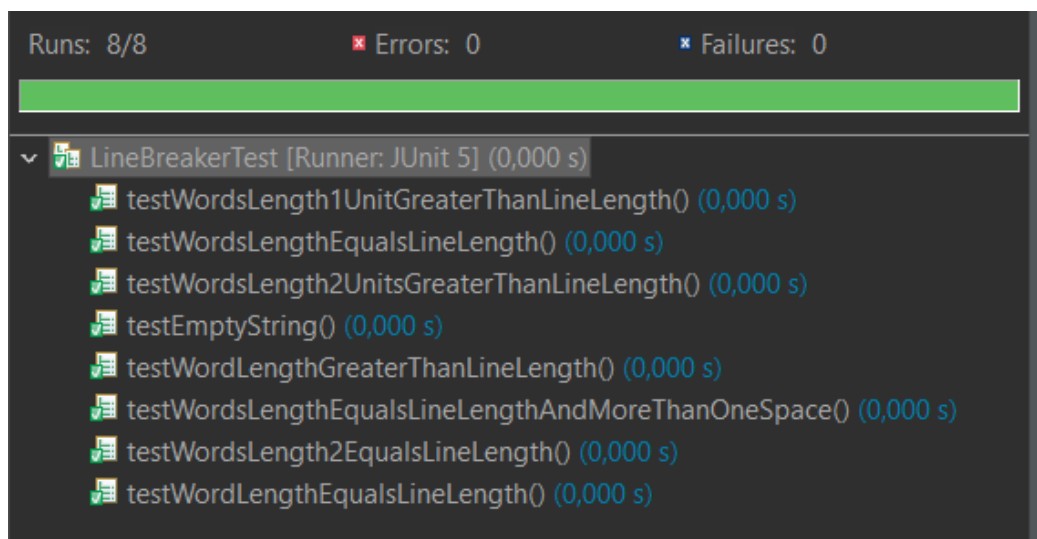
@Test
private void testLineBreaker(String originalText, String expectedText, int lineLength) {
    assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength));
}
```

Comprobamos que el test falla:



Implementamos lo mínimo para que el test no falle:

```
LineBreaker.java x LineBreakerTest.java
1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         if (text.length() <= lineLength){
7             return text;
8         }
9         else if (text.equals("test test")) {
10             return "test\ntest";
11         }
12         else if (text.charAt(lineLength) == ' ') {
13             return text.substring(0, lineLength) + '\n'+
14                 breakText(text.substring(lineLength+1, text.length()), lineLength);
15         }
16         else{
17             return breakText(text, lineLength-1);
18         }
19     }
20 }
```



Refactorizamos y reparamos en que hay que eliminar los **espacios consecutivos** entre dos palabras y sustituirlos por un solo espacio. Asimismo, se deben eliminar los espacios al inicio y final de una descripción. Para ello utilizamos el método **trim**. Adicionalmente, utilizamos el método **replaceAll** el cual sustituye dos o más espacios por uno solo.

```
LineBreaker.java x LineBreakerTest.java
1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         text= text.trim().replaceAll("\\s{2,}", " ");
7         if (text.length() <= lineLength){
8             return text;
9         }
10        else if (text.charAt(lineLength) == ' ') {
11            return text.substring(0, lineLength) + '\n'+
12                breakText(text.substring(lineLength+1, text.length()), lineLength);
13        }
14        else{
15            return breakText(text, lineLength-1);
16        }
17    }
18 }
19 }
```

Finished after 0,11 seconds

Runs: 8/8 ✖ Errors: 0 ✖ Failures: 0

LineBreakerTest [Runner: JUnit 5] (0,000 s)

- ✓ testWordsLength1UnitGreaterThanLineLength() (0,000 s)
- ✓ testWordsLengthEqualsLineLength() (0,000 s)
- ✓ testWordsLength2UnitsGreaterThanLineLength() (0,000 s)
- ✓ testEmptyString() (0,000 s)
- ✓ testWordLengthGreaterThanLineLength() (0,000 s)
- ✓ testWordsLengthEqualsLineLengthAndMoreThanOneSpace() (0,000 s)
- ✓ testWordsLength2EqualsLineLength() (0,000 s)
- ✓ testWordLengthEqualsLineLength() (0,000 s)

9. Noveno ciclo

Creamos el test correspondiente al noveno ciclo:

```

@Test
public void testWordsLength2EqualsLineLength() {
    testLineBreaker("test test test test", "test test\ntest test", 9);
}

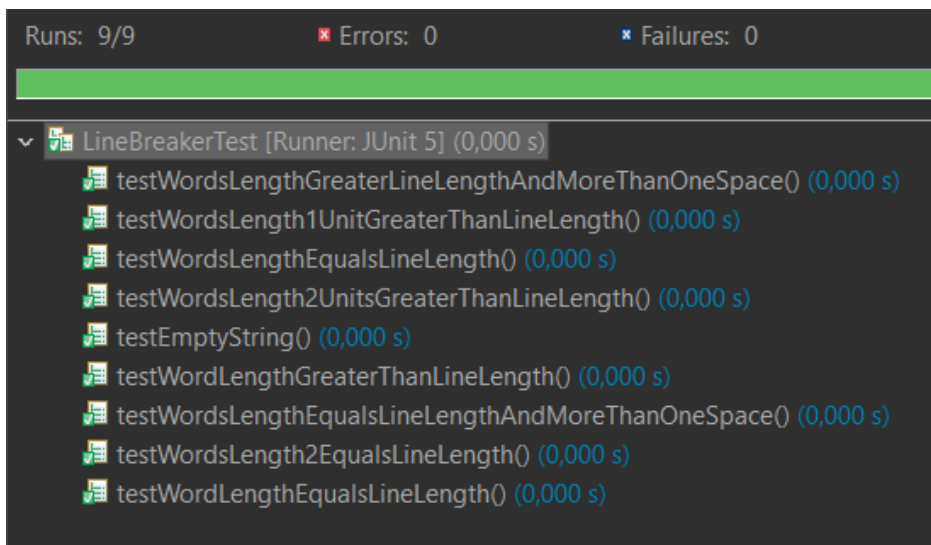
@Test
public void testWordsLengthEqualsLineLengthAndMoreThanOneSpace() {
    testLineBreaker("test  test", "test\ntest", 4);
}

@Test
public void testWordsLengthGreaterLineLengthAndMoreThanOneSpace() {
    testLineBreaker("test  test", "test\ntest", 6);
}

@Test
private void testLineBreaker(String originalText, String expectedText, int lineLength) {
    assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength));
}

```

Al ejecutarlo, pasan todos los tests porque el ciclo anterior es análogo a este:



10. Décimo ciclo

Implementamos el décimo test para el correspondiente ejemplo:

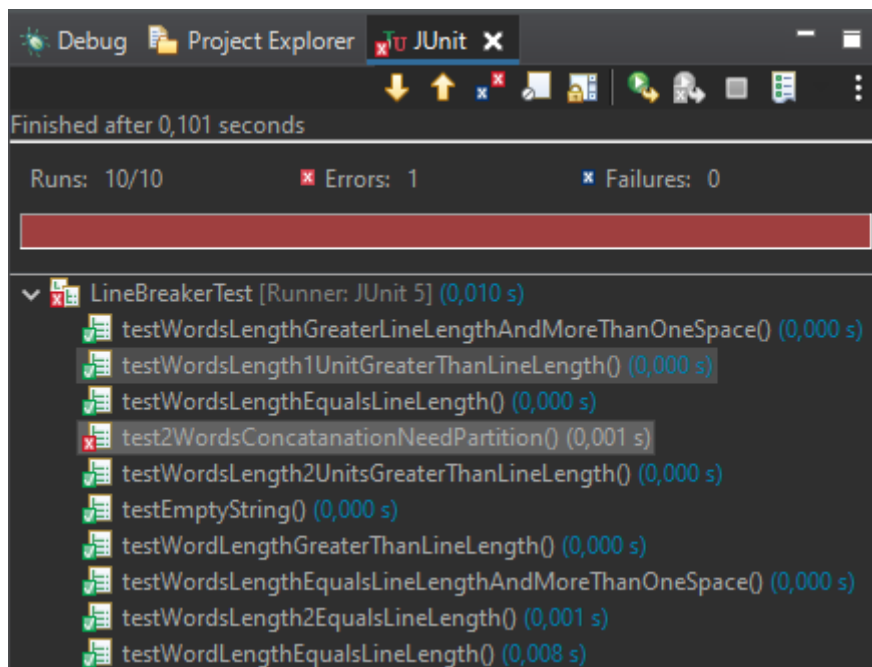
```

@Test
public void test2WordsConcatanationNeedPartition() {
    testLineBreaker("testtest", "test-\ntest", 5);
}

@Test
private void testLineBreaker(String originalText, String expectedText, int lineLength) {
    assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength));
}

```

Observamos que falla:

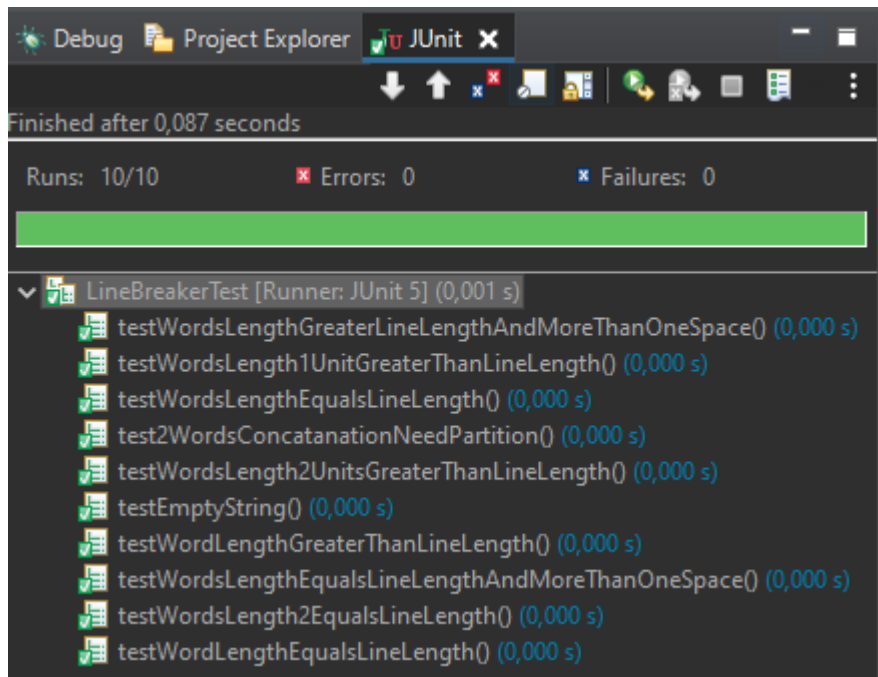


Implementamos el mínimo código para que el test pase:

```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength) {
6         text= text.trim().replaceAll("\\s{2,}", " ");
7         if (text.length() <= lineLength){
8             return text;
9         }
10        else if (text.charAt(lineLength) == ' ') {
11            return text.substring(0, lineLength) + '\n'+
12                breakText(text.substring(lineLength+1, text.length()), lineLength);
13        }
14        else if (text.equals("testtest")) {
15            return "test-\ntest";
16        }
17        else{
18            return breakText(text, lineLength-1);
19        }
20    }

```



Refactorizamos y hacemos el código de calidad:

```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength, int acc) {
6         text= text.trim().replaceAll("\\s{2,}", " ");
7         if (text.length() <= lineLength){
8             return text;
9         }
10        else if (text.charAt(lineLength) == ' ') {
11            return text.substring(0, lineLength) + '\n'+
12                breakText(text.substring(lineLength+1, text.length()), lineLength, acc);
13        }
14        else if (lineLength == 0) {
15            return text.substring(0, acc-1) + "-" + "\n" + breakText(text.substring(acc-1, text.length()), acc, acc);
16        }
17        else{
18            return breakText(text, lineLength-1, acc);
19        }
20    }
21 }

```

Observamos que este caso se resuelve como se muestra en azul:

`breakText("testtest", 5) = "test-\ntest"`

`= text.substring(0, 4) + '-' + '\n' + breakText(text.substring(4, 5), 5) =`

`= "test-\n" + breakText(text.substring(4, 8), 5)=`

`= "test-\n" + breakText("test", 5) =`

`= "test-\ntest"`

Obtuvimos que el caso recursivo por tanto era:

`text.substring(0, lineLength-1) + '-' + '\n' + breakText(substring(lineLength-1, text.length), lineLength)`

Al implementar el código, advertimos que como todo el rato se van a hacer llamadas a `breakText(text, lineLength-1)`:

```
breakText("testtest", 5)
```

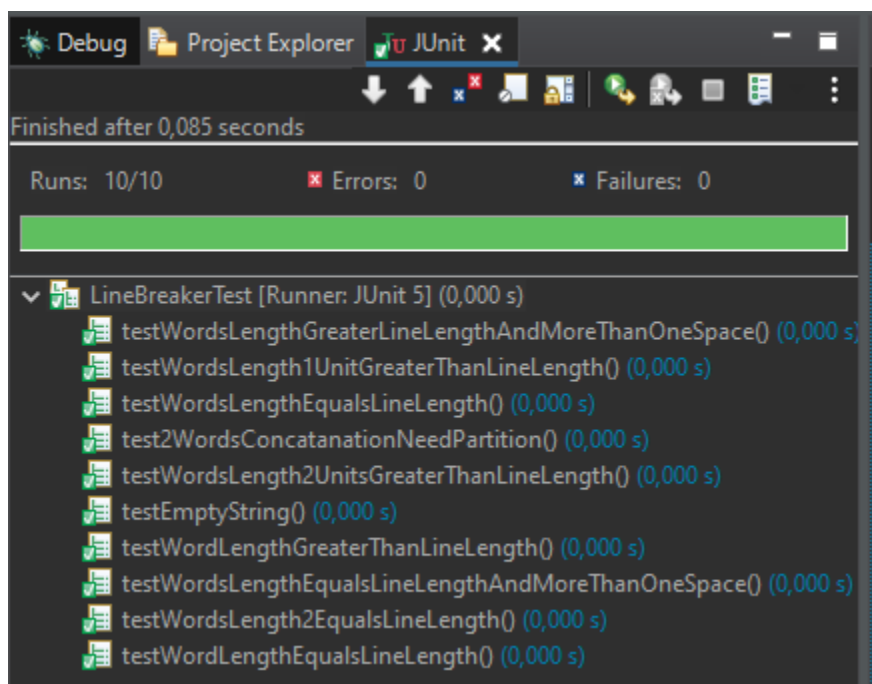
```
breakText("testtest", 4)
```

...

```
breakText("testtest", 0)
```

Se pierde el valor de **lineLength**, por lo tanto, tuvimos que introducir un nuevo parámetro de acumulación que denominamos **acc** (en inglés **accumulator**), que **guarda ese valor**.

Observamos que ya los tests pasan:



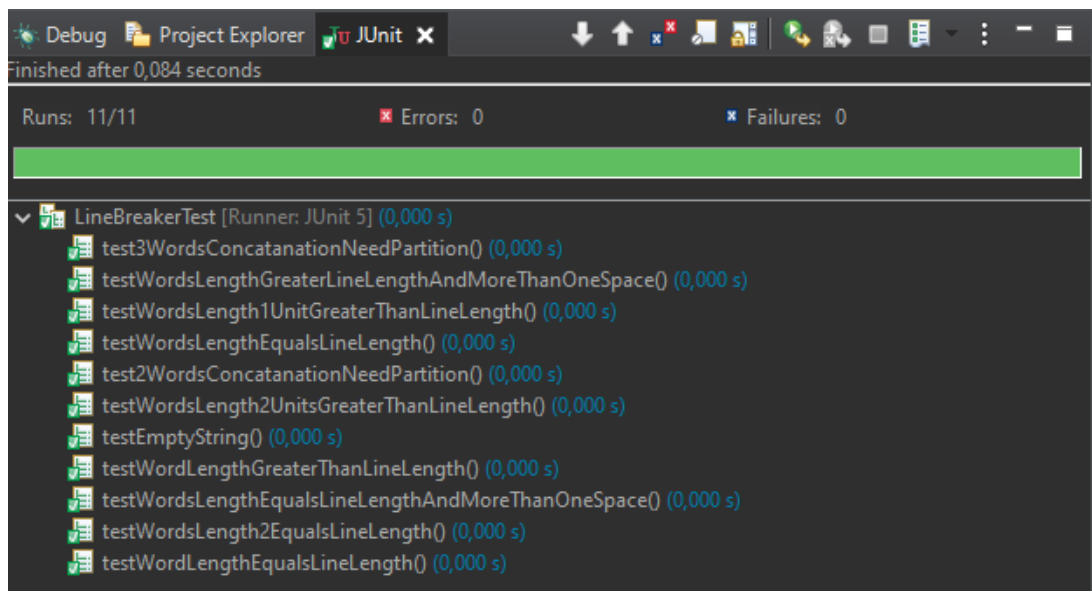
11. Onceavo ciclo

Creamos el test para este ciclo:

```
@Test
public void test3WordsConcatanationNeedPartition() {
    testLineBreaker("testtesttest", "test-\ntest-\ntest", 5);
}

@Test
private void testLineBreaker(String originalText, String expectedText, int lineLength) {
    assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength, lineLength));
}
```

Los test pasan ya que el algoritmo que hemos implementado para el caso anterior resuelve este caso:



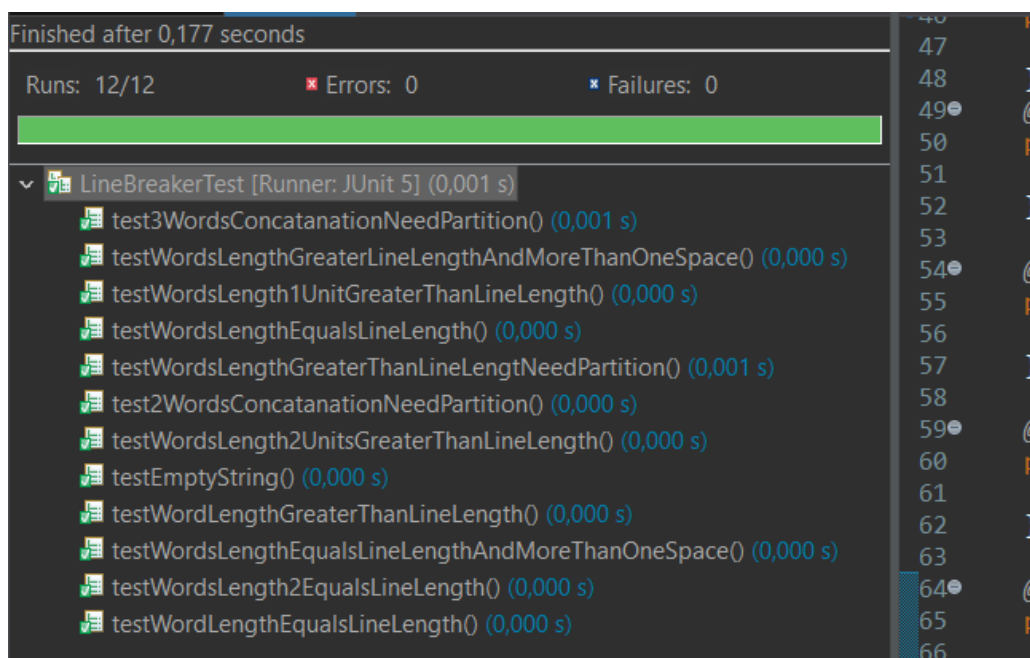
12. Doceavo ciclo

Implementamos el test correspondiente al ciclo doceavo:

```
@Test
public void testWordsLengthGreaterthanLineLengtNeedPartition() {
    testLineBreaker("test test", "te-\nst\nte-\nst", 3);
}

@Test
private void testLineBreaker(String originalText, String expectedText, int lineLength) {
    assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength, lineLength));
}
```

Al ejecutar los tests vemos que no fallan, ya que el algoritmo implementado cubre todos los casos hasta el momento.



Llegados a este punto creemos que ya hemos implementado el algoritmo completo pero se seguirá comprobando en los siguientes ciclos del TDD.

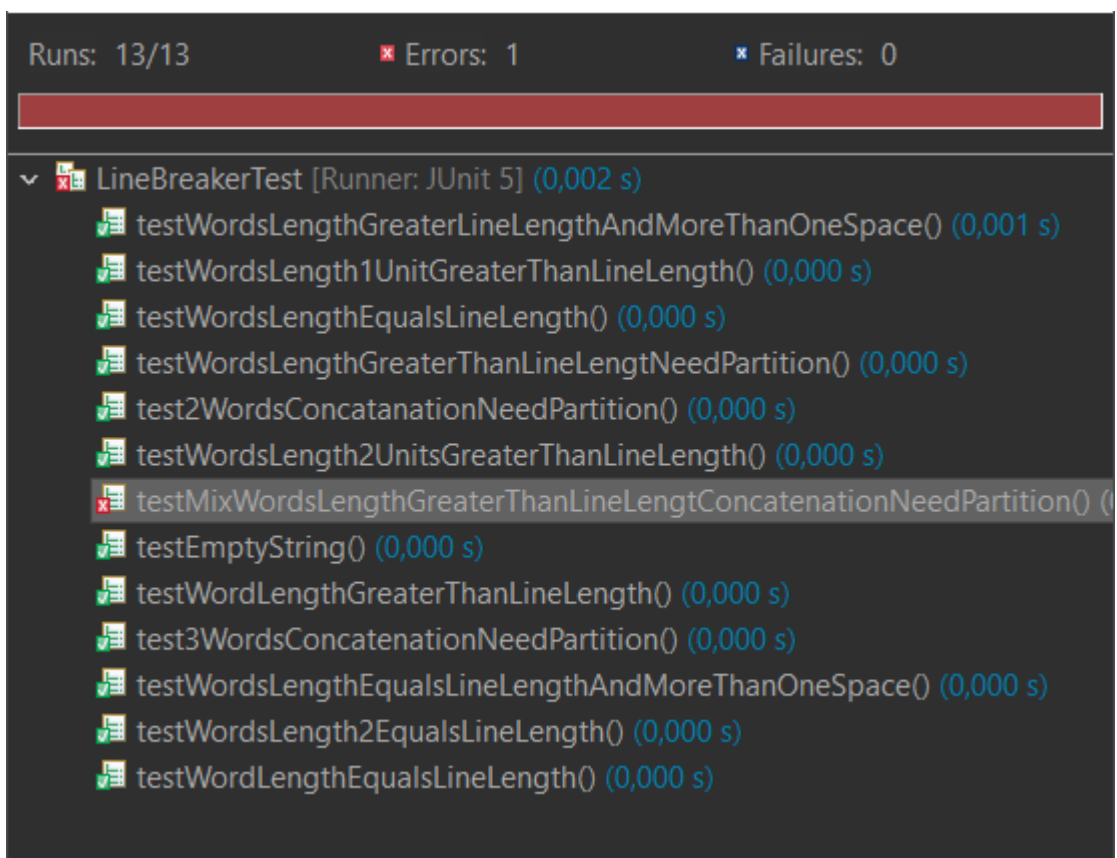
13. Treceavo ciclo

Aplicamos el test correspondiente a este ciclo:

```
    @Test
    public void testMixWordsLengthGreaterThanLineLengtConcatenationNeedPartition() {
        testLineBreaker("test 1234567 test", "test\n12345-\n67\ntest", 6);
    }

    @Test
    private void testLineBreaker(String originalText, String expectedText, int lineLength) {
        assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength, lineLength));
    }
```

Al ejecutar los tests nos dimos cuenta de que éste fallaba, así que creímos que el algoritmo cubría todos los casos:



Analizamos nuestro algoritmo anterior y comprendimos que no habíamos sustituido **lineLength** por el **acumulador** en la llamada a la función, por lo que todo el rato se hacía la división con la primera palabra dividida.

```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength, int acc) {
6         text= text.trim().replaceAll("\s{2,}", " ");
7         if (text.length() <= lineLength){
8             return text;
9         }
10        else if (text.charAt(lineLength) == ' ') {
11            return text.substring(0, lineLength) + '\n'+
12                breakText(text.substring(lineLength+1, text.length()), lineLength, acc);
13        }
14        else if (lineLength == 0) {
15            return text.substring(0, acc-1) + "-" + "\n" + breakText(text.substring(acc-1, text.length()), acc, acc);
16        }
17        else{
18            return breakText(text, lineLength-1, acc);
19        }
20    }
21 }
22 }
23

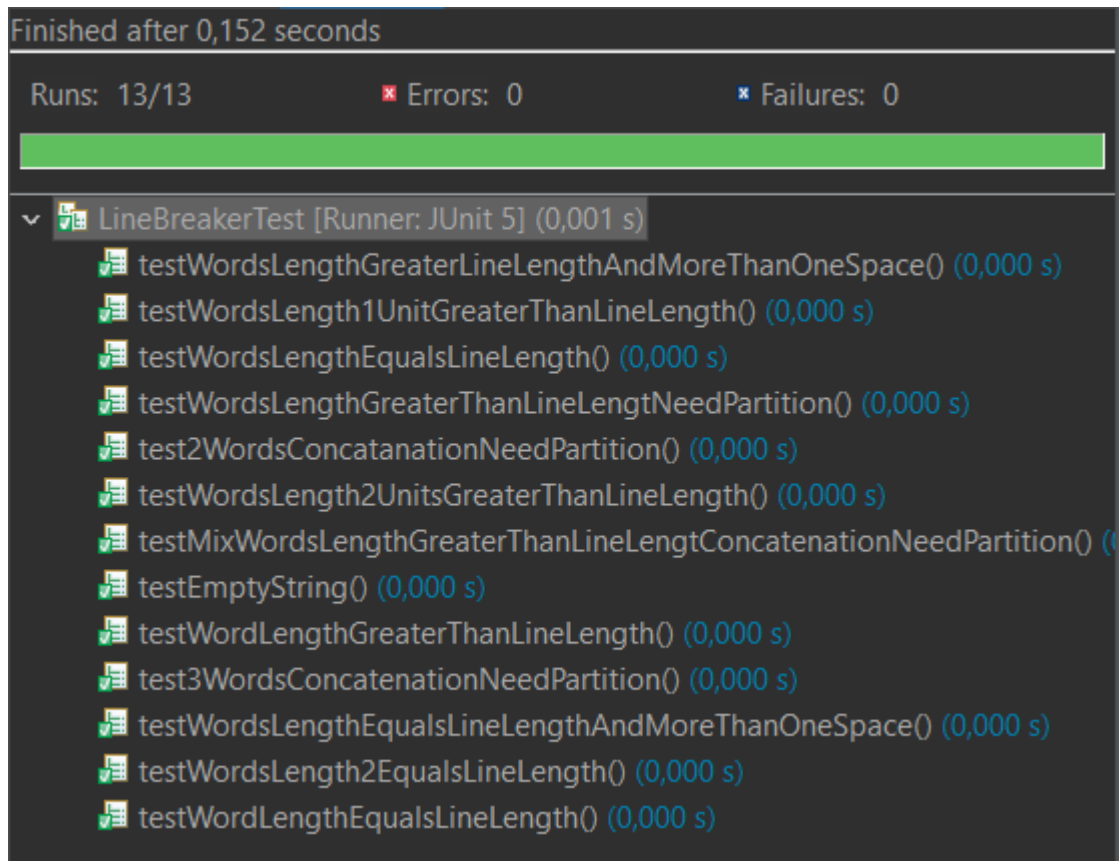
```

```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength, int acc) {
6         text= text.trim().replaceAll("\s{2,}", " ");
7         if (text.length() <= lineLength){
8             return text;
9         }
10        else if (text.charAt(lineLength) == ' ') {
11            return text.substring(0, lineLength) + '\n'+
12                breakText(text.substring(lineLength+1, text.length()), acc, acc);
13        }
14        else if (lineLength == 0) {
15            return text.substring(0, acc-1) + "-" + "\n" + breakText(text.substring(acc-1, text.length()), acc, acc);
16        }
17        else{
18            return breakText(text, lineLength-1, acc);
19        }
20    }
21 }
22 }
23

```

Después ejecutamos los test y hemos observado que no fallaban:



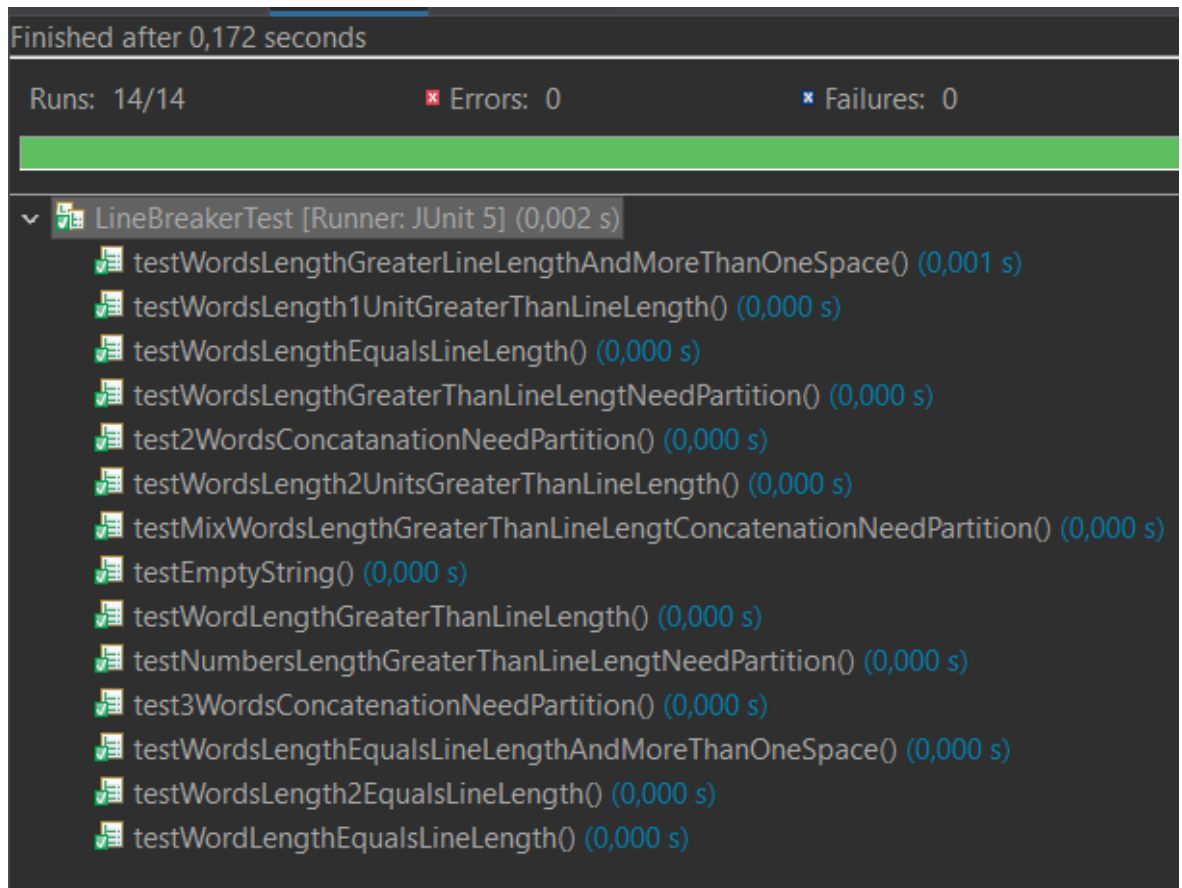
14. Catorceavo ciclo

Implementamos el código del test del ciclo correspondiente:

```
@Test
public void testNumbersLengthGreaterThanLineLengthNeedPartition() {
    testLineBreaker("123456789", "12-\n34-\n56-\n789", 3);
}

@Test
private void testLineBreaker(String originalText, String expectedText, int lineLength) {
    assertEquals(expectedText, new LineBreaker().breakText(originalText, lineLength, lineLength));
}
```

Ejecutamos los tests y vemos que pasan sin ningún fallo:



Finalmente concluimos que el algoritmo implementado es válido para todos los posibles casos.

```
LineBreaker.java x LineBreakerTest.java
1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public String breakText(String text, int lineLength, int acc) {
6         text= text.trim().replaceAll("\\s{2,}", " ");
7         if (text.length() <= lineLength){
8             return text;
9         }
10        else if (text.charAt(lineLength) == ' ') {
11            return text.substring(0, lineLength) + '\n'+
12                breakText(text.substring(lineLength+1, text.length()), acc, acc);
13        }
14        else if (lineLength == 0) {
15            return text.substring(0, acc-1) + "- " + "\n" + breakText(text.substring(acc-1, text.length()), acc, acc);
16        }
17        else{
18            return breakText(text, lineLength-1, acc);
19        }
20    }
21 }
22 }
23 }
```

15. Arreglos

Una vez completados los catorce ciclos, invocamos al método **breakLine** en **BookService** para hacer la división de líneas de la descripción de todos los libros que se guardaban en la base de datos:

```

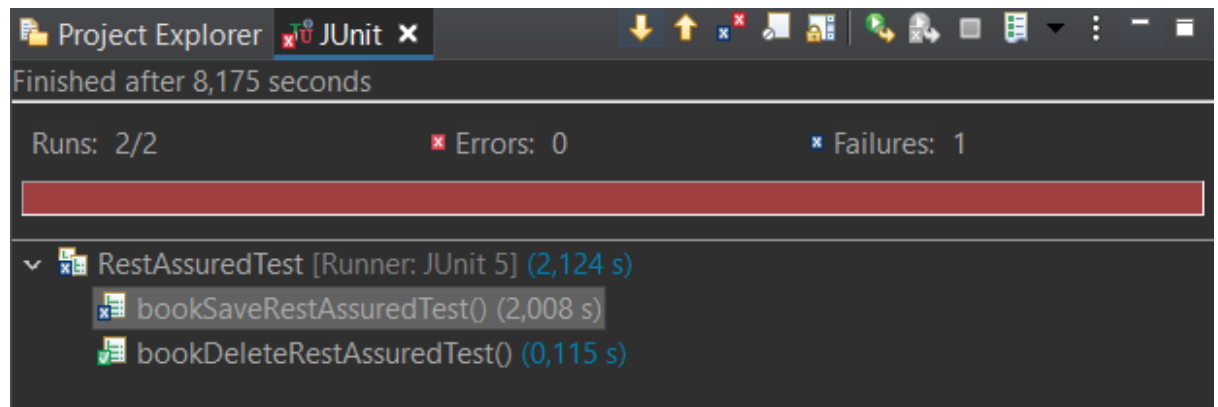
public Book save(Book book) {
    book.setDescription(LineBreaker.breakText(book.getDescription(),10, 10));

    Book newBook = repository.save(book);
    notificationService.notify("Book Event: book with title="+newBook.getTitle()+" was created");
    return newBook;
}

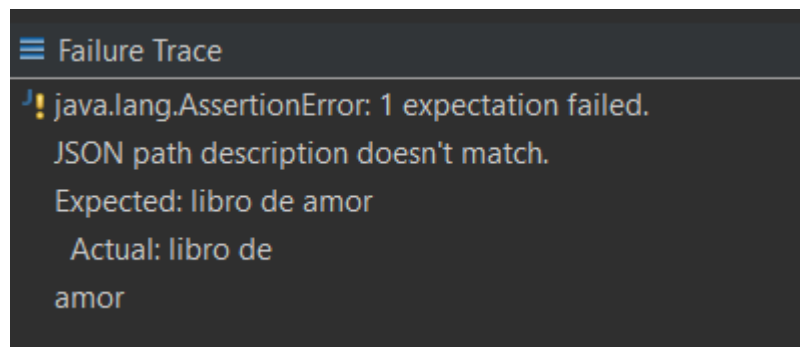
```

A continuación, ejecutamos todos los test (nuevos y antiguos) para comprobar si siguen pasando.

El método **bookSaveRestAssuredTest** de la clase **RestAssuredTest** falla:



Se esperaba "libro de\n amor", debido al método **breakLine** que se aplica sobre la descripción:



Hemos tenido que cambiar el then:

```

@Test
void bookSaveRestAssuredTest() {

    // Given

    Response response = given().
        contentType("application/json").
        body("{\"title\":\"cumbres borrascosas\",\"description\":\"libro de amor\"}").

    // When

    when().
        post("/api/books/").thenReturn();

    Integer id = from(response.getBody().asString()).get("id");

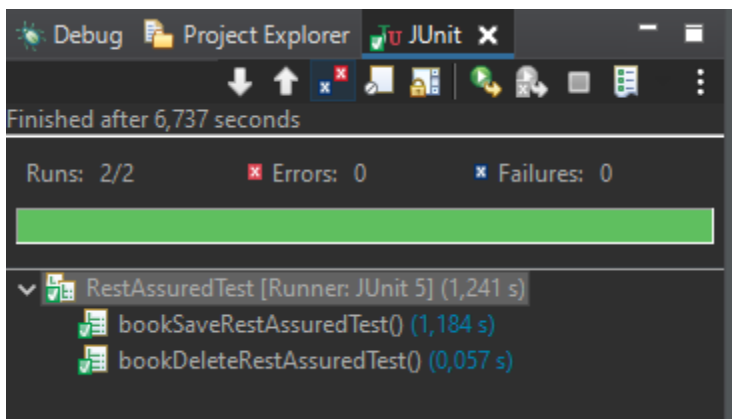
    // Then

    when().
        get("api/books/{id}", id).
    then()
        .statusCode(200).
        body("title", equalTo("cumbres borrascosas")).
        body("description", equalTo("libro de\namor"));

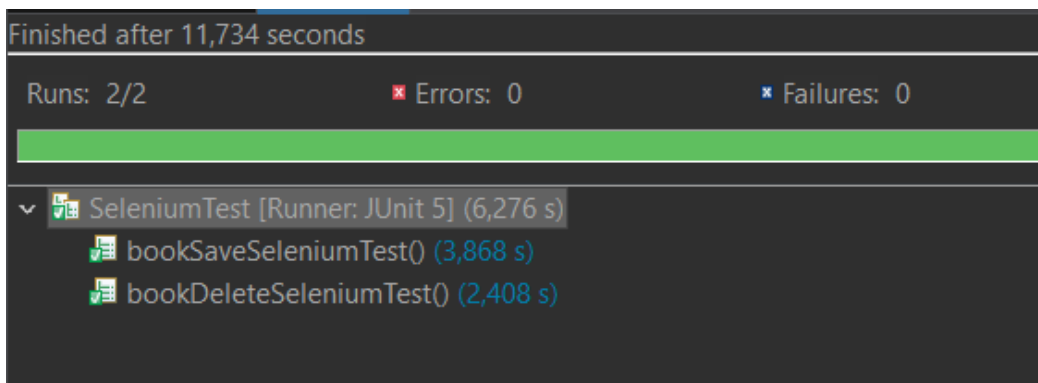
}

```

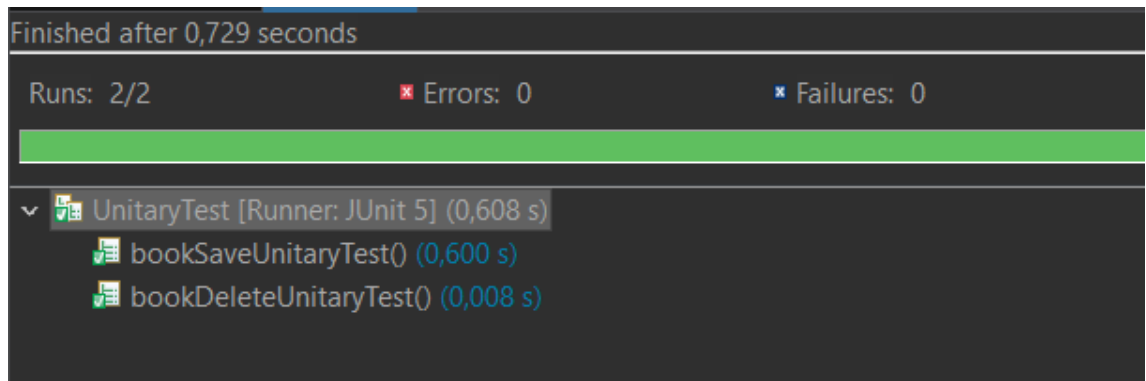
Y comprobamos que ya sí pasan todos los tests de esta clase:



Los tests de la clase **SeleniumTest** no fallan:



Tampoco los de **UnitaryTest**:



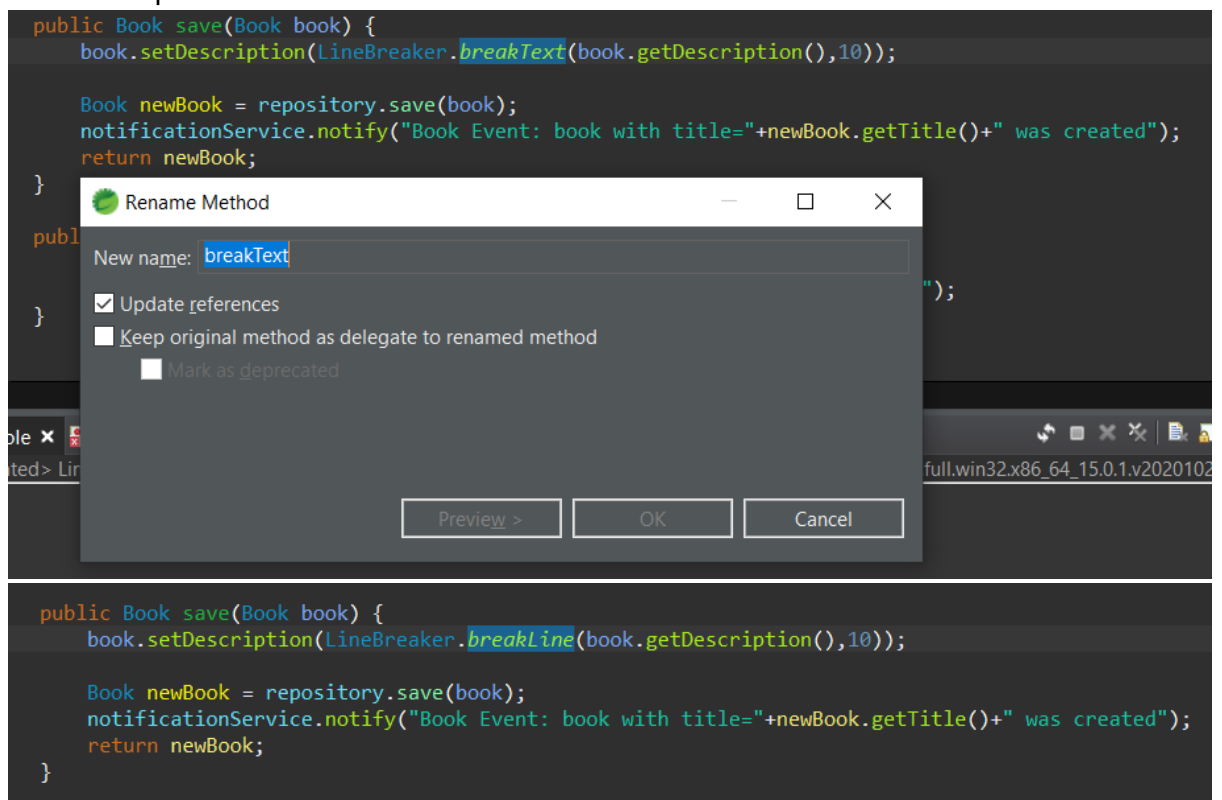
Al revisar los tests, nos hemos percatado de un error de sintaxis en el nombre del test del ciclo 10:

```
@Test
public void test2WordsConcatanationNeedPartition() {
    testLineBreaker("testtest", "test-\ntest", 5);
}
```

La corrección es:

```
@Test
public void test2WordsConcatenationNeedPartition() {
    testLineBreaker("testtest", "test-\ntest", 5);
}
```

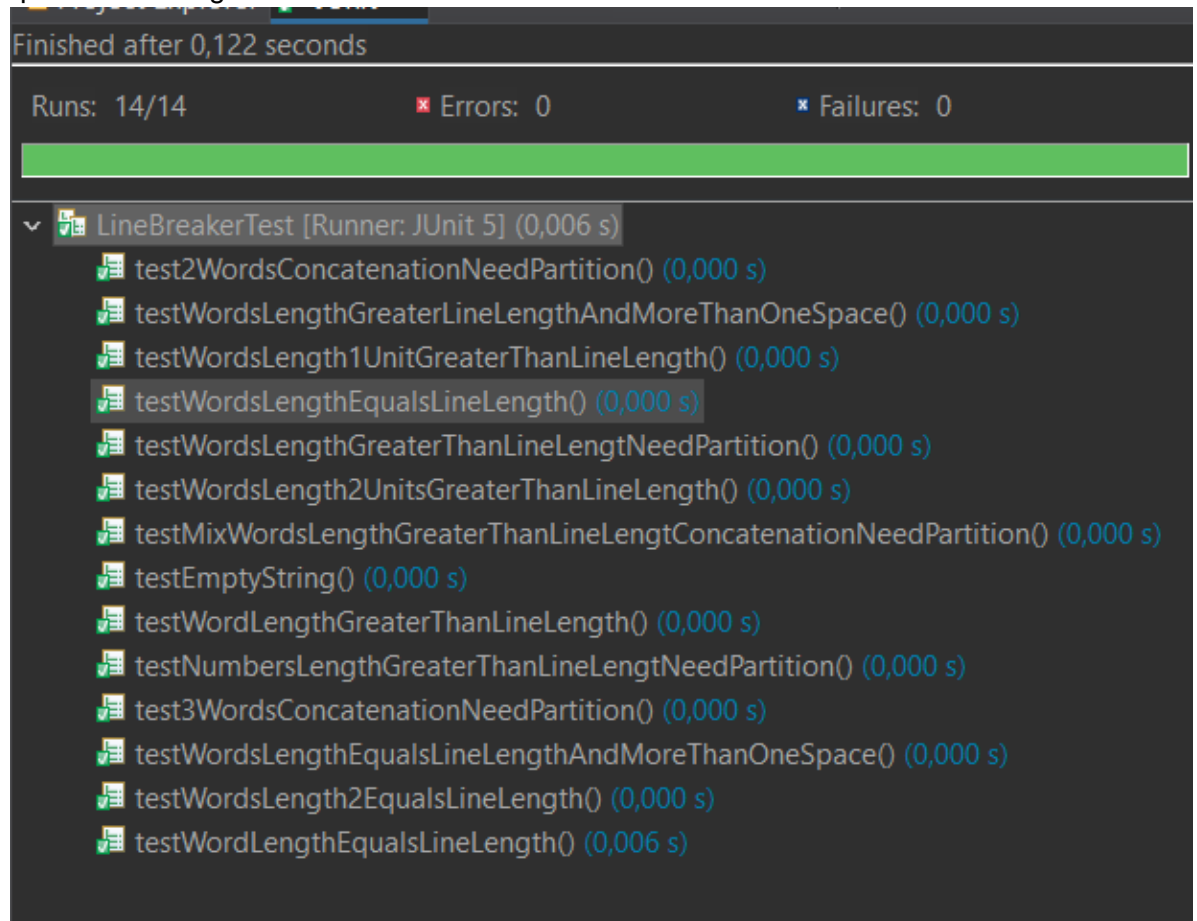
Tras haber leído la quinta versión de la práctica tuvimos que renombrar el método de **breakText** por **breakLine** como indicaba el enunciado:



Después de haber revisado con detenimiento los requisitos de la práctica, nos fijamos en que no se podía cambiar la **signatura del método**, es decir, el **número de parámetros** del mismo. Ante esta situación se ha optado por crear una **nueva función auxiliar** que denominamos **breakLineAux** la cual porta los argumentos necesarios para cumplir las distintas funcionalidades exigidas.

```
1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4     public static String breakLine(String text, int lineLength) {
5         return breakLineAux(text, lineLength, lineLength);
6     }
7     public static String breakLineAux(String text, int lineLength, int acc) {
8         text= text.trim().replaceAll("\\s{2,}", " ");
9         if (text.length() <= lineLength){
10             return text;
11         }
12         else if (text.charAt(lineLength) == ' ') {
13             return text.substring(0, lineLength) + '\n'+
14                 breakLineAux(text.substring(lineLength+1, text.length()), acc, acc);
15         }
16         else if (lineLength == 0) {
17             return text.substring(0, acc-1) + "-" + "\n" + breakLineAux(text.substring(acc-1, text.length()), acc, acc);
18         }
19         else{
20             return breakLineAux(text, lineLength-1, acc);
21         }
22     }
23 }
```

Pasamos los test para ver si todo sigue funcionando correctamente y observamos que no falla ninguno:



Una de las condiciones que se requerían era que la longitud de las líneas no podía ser inferior a dos, por lo tanto se ha implementado el test que contempla que se lanza una excepción en caso de que **lineLength** sea inferior a dos.

```

@Test
public void testLineLengthLessThan2throwException() {
    assertThrows(RuntimeException.class, ()->{
        new LineBreaker().breakLine("", 1);
    });
}

```

Observamos que el test falla:

Runs: 15/15 ✖ Errors: 0 ✖ Failures: 1

LineBreakerTest [Runner: JUnit 5] (0,008 s)

- test2WordsConcatenationNeedPartition() (0,000 s)
- testWordsLengthGreaterLineLengthAndMoreThanOneSpace() (0,000 s)
- testWordsLength1UnitGreaterThanLineLength() (0,000 s)
- ✖ testLineLengthLessThan2throwException() (0,001 s)
- testWordsLengthEqualsLineLength() (0,000 s)
- testWordsLengthGreaterThanLineLengthNeedPartition() (0,000 s)
- testWordsLength2UnitsGreaterThanLineLength() (0,000 s)
- testMixWordsLengthGreaterThanLineLengthConcatenationNeedPartition() (0,000 s)
- testEmptyString() (0,000 s)
- testWordLengthGreaterThanLineLength() (0,000 s)
- testNumbersLengthGreaterThanLineLengthNeedPartition() (0,000 s)
- test3WordsConcatenationNeedPartition() (0,000 s)
- testWordsLengthEqualsLineLengthAndMoreThanOneSpace() (0,000 s)
- testWordsLength2EqualsLineLength() (0,000 s)
- testWordLengthEqualsLineLength() (0,007 s)

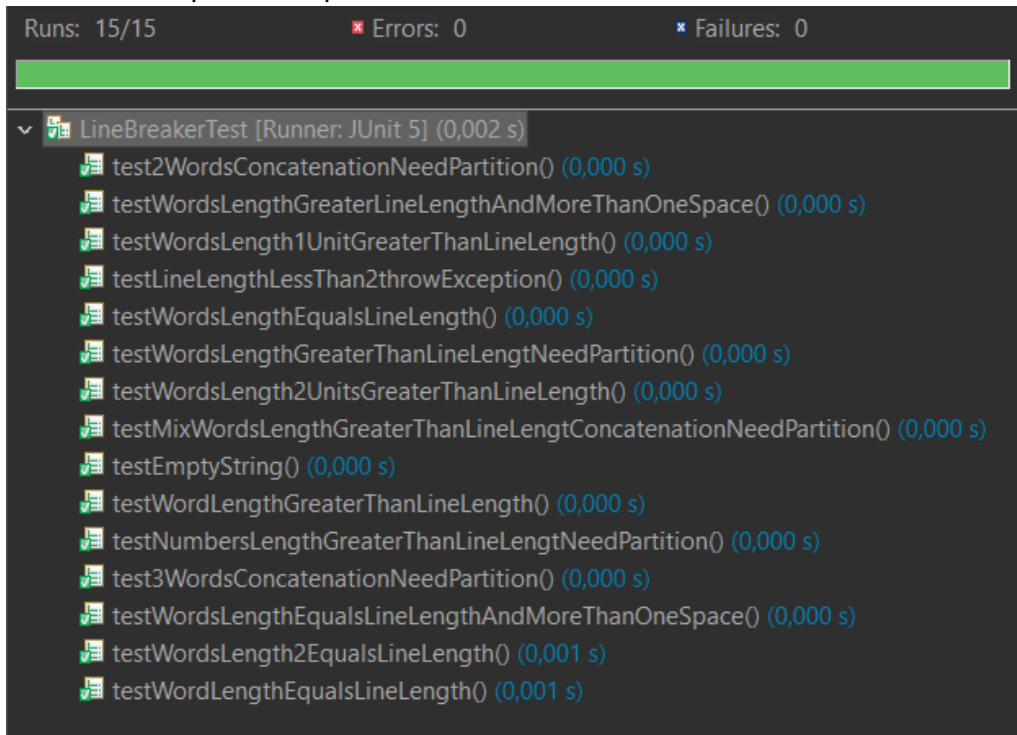
Implementamos el código mínimo e indispensable para que el test no falle:

```

LineBreaker.java LineBreakerTest.java BookService.java RestAssuredTest.java SeleniumTest.java
1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4     public static String breakLine(String text, int lineLength) {
5         if (lineLength < 2) {
6             throw new RuntimeException("El tamaño de línea no puede ser inferior a 2");
7         }
8         return breakLineAux(text, lineLength, lineLength);
9     }
10    public static String breakLineAux(String text, int lineLength, int acc) {
11        text = text.trim().replaceAll("\\s{2,}", " ");
12        if (text.length() <= lineLength) {
13            return text;
14        }
15        else if (text.charAt(lineLength) == ' ') {
16            return text.substring(0, lineLength) + '\n' +
17                breakLineAux(text.substring(lineLength+1, text.length()), acc, acc);
18        }
19        else if (lineLength == 0) {
20            return text.substring(0, acc-1) + "- " + "\n" + breakLineAux(text.substring(acc-1, text.length()), acc, acc);
21        }
22        else {
23            return breakLineAux(text, lineLength-1, acc);
24        }
25    }
26 }

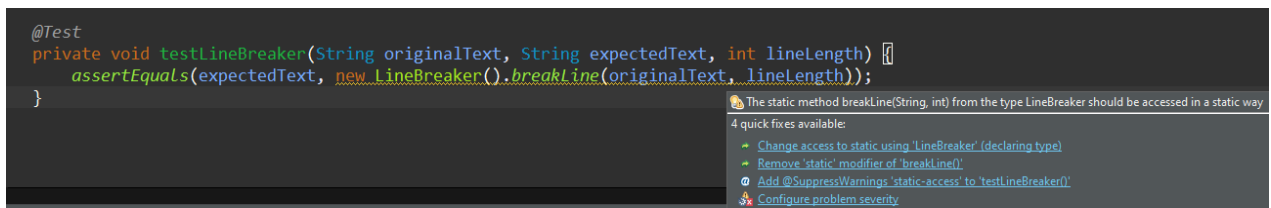
```

Observamos que el test pasa:

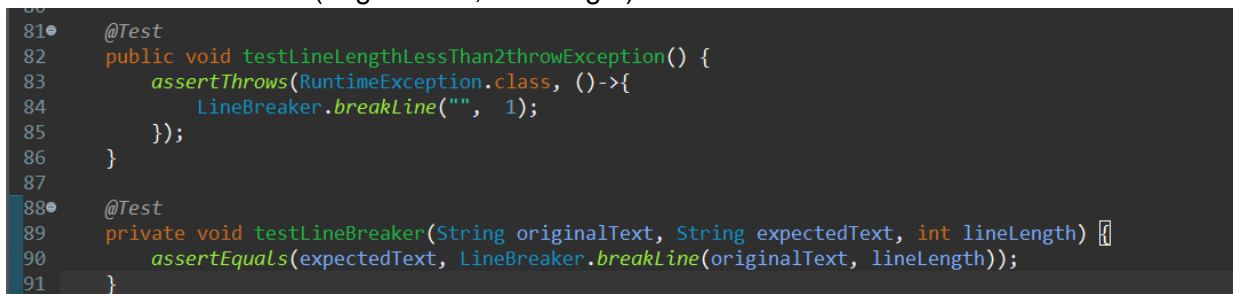


El código ya está refactorizado, ya que no hay otra forma de lanzar la excepción y dejarlo más compacto.

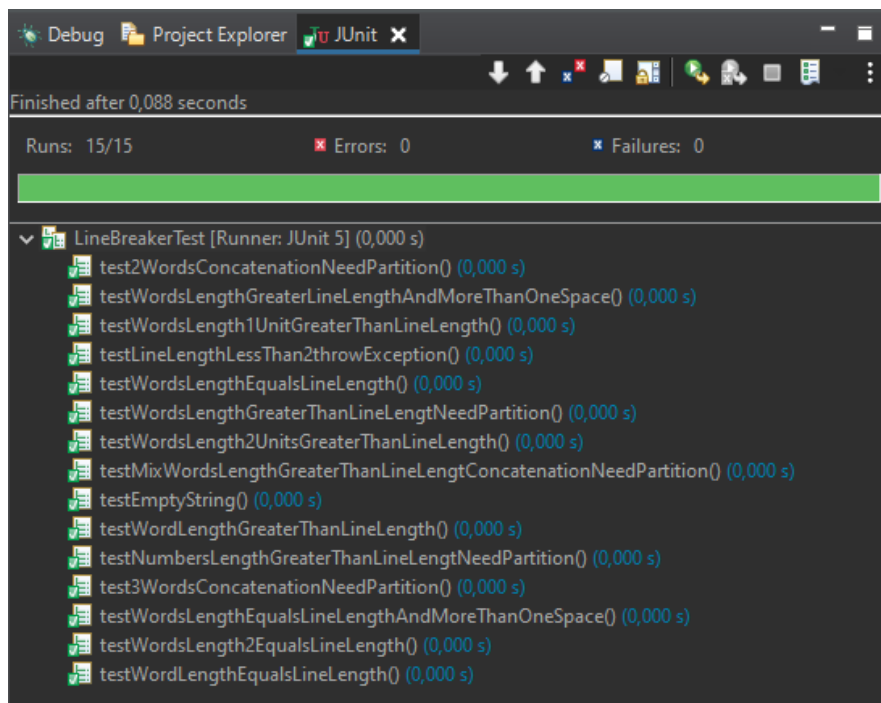
Por otro lado, hemos suprimido los **warnings** dentro de la clase **LineBreakerTest** que surgían porque se debería haber accedido al método estático **breakLine** de manera estática:



Para arreglarlo, invocamos al método estático de la forma convencional con **LineBreaker.breakLine(originalText, lineLength)**:



Seguimos observando que siguen pasando los tests:



El código final es el que sigue:

```

1 package es.urjc.code.daw.library.book;
2
3 public class LineBreaker {
4
5     public static String breakLine(String text, int lineLength) {
6         if (lineLength < 2) {
7             throw new RuntimeException("El tamaño de línea no puede ser inferior a 2");
8         }
9         return breakLineAux(text, lineLength, lineLength);
10    }
11
12    public static String breakLineAux(String text, int lineLength, int acc) {
13        text= text.trim().replaceAll("\\s{2,}", " ");
14        if (text.length() <= lineLength){
15            return text;
16        }
17        else if (text.charAt(lineLength) == ' ') {
18            return text.substring(0, lineLength) + '\n'+
19                breakLineAux(text.substring(lineLength+1, text.length()), acc, acc);
20        }
21        else if (lineLength == 0) {
22            return text.substring(0, acc-1) + "- " + "\n" + breakLineAux(text.substring(acc-1, text.length()), acc, acc);
23        }
24        else{
25            return breakLineAux(text, lineLength-1, acc);
26        }
27    }
28 }

```