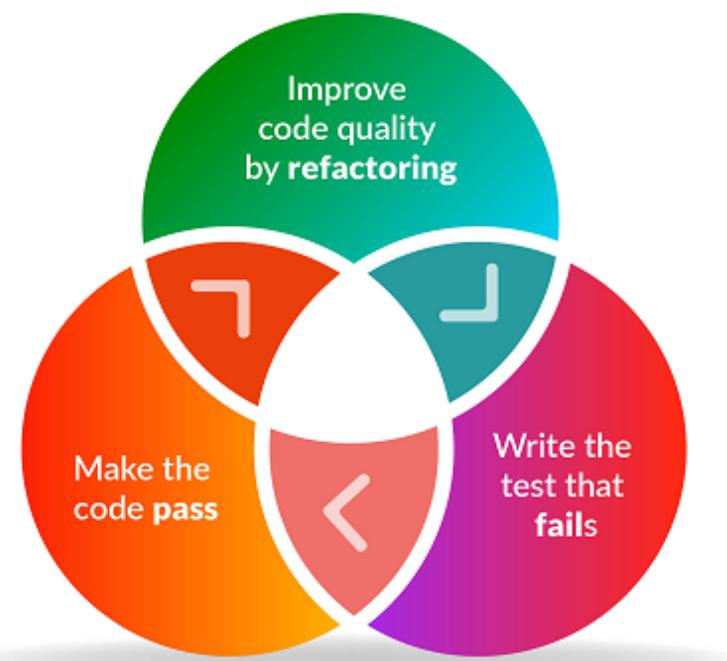


# Práctica 2 – Test Driven Development



Ampliación de Ingeniería del Software – 3º 2020-2021

Autores: Juan Antonio Vinaches Vizcaíno y Sonia Casero Pérez

# Ciclo 1

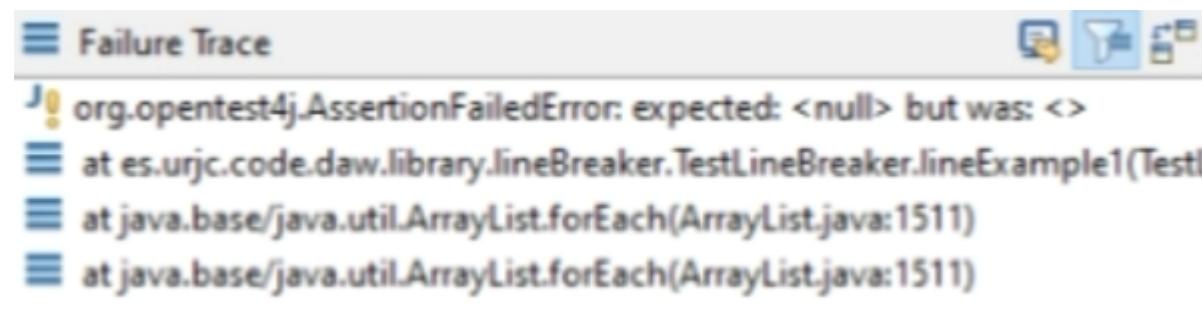
En este primer ciclo se comprueba que el método **breakText()** es capaz de procesar cadenas vacías.

## Código del test

```
@Test  
void lineExample1() {  
    assertEquals("", LineBreaker.breakText("", 2));  
}
```

## Mensaje de error obtenido al fallar el test

Al ejecutar el test por primera vez, como no se ha implementado el método **breakText()** para que procese cadenas vacías, devuelve **null**.



## Implementación que hace pasar el test

Para hacer pasar el test, hacemos que se devuelva el resultado esperado al procesar una cadena vacía.

```
public class LineBreaker {  
    public String breakText(String text, int lineLength) {  
        return "";  
    }  
}
```

## Implementación después de refactorizar

La refactorización se corresponde con la implementación al ser el primer ejemplo de prueba.

```
public class LineBreaker {  
    public String breakText(String text, int lineLength) {  
        return "";  
    }  
}
```

## Ciclo 2

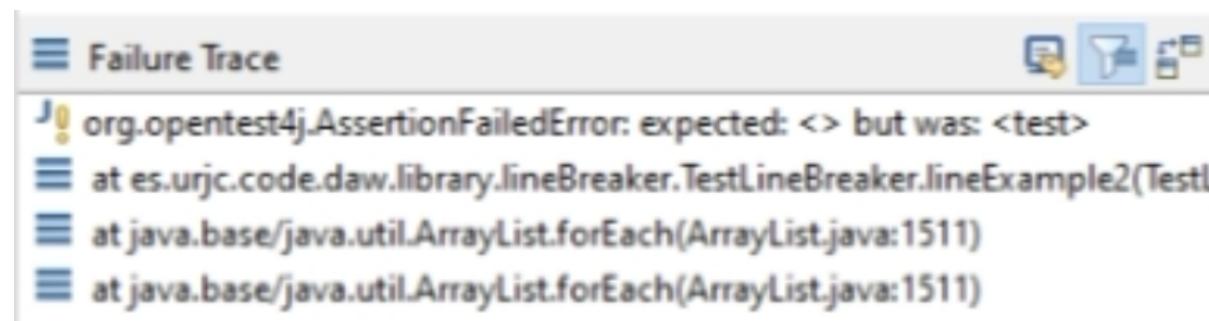
### Código del test

En este ejemplo se comprueba que se muestra bien una línea de texto que cabe en la longitud de línea dada.

```
@Test  
void lineExample2() {  
    assertEquals("test", LineBreaker.breakText("test", 4));  
}
```

### Mensaje de error obtenido al fallar el test

Falla el test porque implementamos anteriormente que **breakText()** devolviera cadena vacía.



## Implementación que hace pasar el test

Para solventar el problema devolvemos directamente el texto de entrada.

```
public class LineBreaker {  
    public String breakText(String text, int lineLength) {  
        return text;  
    }  
}
```

## Implementación después de refactorizar

No fue necesario refactorizar.

## Ciclo 3

El siguiente ciclo trata de comprobar si el método **breakText()** procesa cadenas de texto cuya longitud sea inferior a la longitud introducida por parámetro.

### Código del test

```
@Test  
void lineExample3() {  
    assertEquals(lineBreaker.breakText("test", 5), "test");  
}
```

### Mensaje de error obtenido al fallar el test

No hubo ningún fallo. Pasó el test.

## Implementación que hace pasar el test

La implementación se corresponde con el código del ciclo anterior.

## Implementación después de refactorizar

La implementación se corresponde con el código del ciclo anterior.

## Ciclo 4

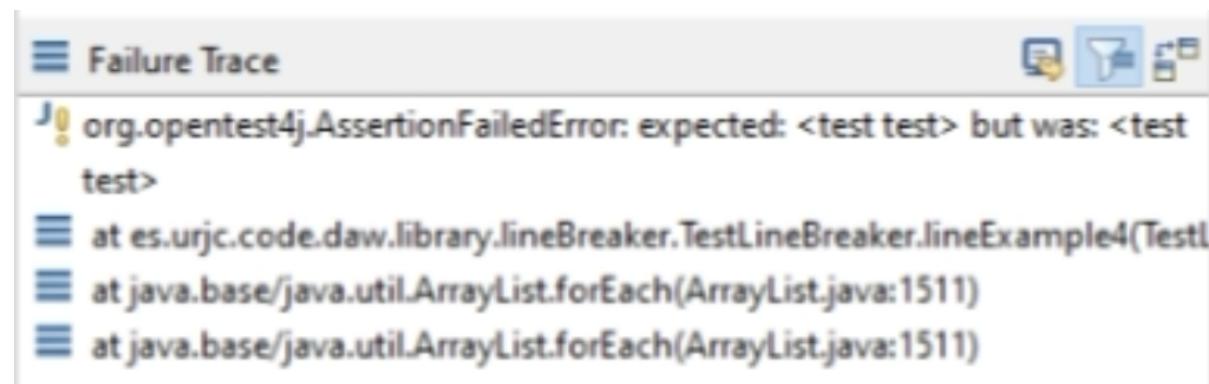
### Código del test

En este test se comprueba si se divide la cadena de entrada según la longitud máxima de línea especificada.

```
@Test  
void lineExample4() {  
    assertEquals(lineBreaker.breakText("test test", 4), "test\ntest");  
}
```

### Mensaje de error obtenido al fallar el test

Falla debido a que **breakText()** devuelve el texto de entrada sin procesarlo.



### Implementación que hace pasar el test

Introdujimos código que sustituía el carácter en el índice 4 (el espacio) por un salto de línea.

```
public class LineBreaker {  
    public String breakText(String text, int lineLength) {  
        if(text.length()> lineLength) {  
            StringBuilder textAux = new StringBuilder(text);  
            textAux.setCharAt(4, '\n');  
            text = textAux.toString();  
        }  
        return text;  
    }  
}
```

## Implementación después de refactorizar

Lo refactorizamos para que en vez de sustituir en el índice 4, sustituya en el índice equivalente a la longitud de línea.

```
public final char NEXTLINE = '\n';
public String breakText(String text, int lineLength) {
    if(text.length() > lineLength) {
        StringBuilder textAux = new StringBuilder(text);
        textAux.setCharAt(lineLength, NEXTLINE);
        text = textAux.toString();
    }
    return text;
}
```

## Ciclo 5

En este ciclo se comprueba que **breakText()** introduce saltos de línea dada una longitud de línea mayor que la anterior.

### Código del test

```
@Test
void lineExample5() {
    assertEquals(lineBreaker.breakText("test test", lineLength: 5), "test\ntest");
}
```

### Mensaje de error obtenido al fallar el test

El test falla debido a que se sustituye el carácter donde se produce el corte de línea sin tener en cuenta el carácter que está siendo sustituido. En este caso, una ‘t’.

```
JUnit org.opentest4j.AssertionFailedError: expected: <test
est> but was: <test
test>
  at es.urjc.code.daw.library.lineBreaker.TestLineBreaker.lineExample5(TestLineBreaker.java:38)
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

## Implementación que hace pasar el test

Para pasar el test, conocemos el resultado esperado y sustituimos el carácter donde está el espacio por el salto de línea: `textAux.setCharAt(4,NEXTLINE)`.

```
public class LineBreaker {  
    public final char NEXTLINE = '\n';  
    public String breakText(String text, int lineLength) {  
        if(text.length() > lineLength) {  
            StringBuilder textAux = new StringBuilder(text);  
            textAux.setCharAt(4, NEXTLINE);  
            text = textAux.toString();  
        }  
        return text;  
    }  
}
```

## Implementación después de refactorizar

Al refactorizar, hemos creado el método `searchNearestSpace()` para poder encontrar los índices de los espacios más cercanos y así, poder sustituirlos por saltos de línea.

```
public class LineBreaker {  
    public static final char NEXTLINE = '\n';  
    public String breakText(String text, int lineLength) {  
        if(text.length() <= lineLength) {return text;}  
        StringBuilder textAux = new StringBuilder(text);  
        int index = searchNearestSpace(textAux, lineLength);  
        textAux.setCharAt(index, NEXTLINE);  
        text = textAux.toString();  
        return text;  
    }  
    private int searchNearestSpace(  
        StringBuilder text, int startingIndex  
    ) {  
        int index = startingIndex;  
        char charAtIndex = text.charAt(index);  
        while (index >= 0 && charAtIndex != ' ') {  
            index--;  
            charAtIndex = text.charAt(index);  
        }  
        return index;  
    }  
}
```

# Ciclo 6

## Código del test

En este test se comprueba que el salto de línea se introduce en el espacio a pesar de tener una longitud de línea mayor al tamaño de la última palabra de la línea.

```
@Test  
void lineExample6() {  
    assertEquals(lineBreaker.breakText("test test", 6), "test\ntest");  
}
```

## Mensaje de error obtenido al fallar el test

No hubo ningún fallo. Se pasó el test

## Implementación que hace pasar el test

La implementación se corresponde con el código del ciclo anterior.

## Implementación después de refactorizar

Se compactaron las dos líneas finales de **breakLine()** en una. Antes se devolvía **text** asignándole previamente el valor de **textAux.toString()**. Ahora simplemente devolvemos dicho valor.

```
public class LineBreaker {  
    public static final char NEXTLINE = '\n';  
    public String breakText(String text, int lineLength) {  
        if(text.length() <= lineLength) {return text;}  
        StringBuilder textAux = new StringBuilder(text);  
        int index = searchNearestSpace(textAux, lineLength);  
        textAux.setCharAt(index, NEXTLINE);  
        return textAux.toString();  
    }  
    private int searchNearestSpace(  
        StringBuilder text, int startingIndex  
    ) {  
        int index = startingIndex;  
        char charAtIndex = text.charAt(index);  
        while (index >= 0 && charAtIndex != ' ') {  
            index--;  
            charAtIndex = text.charAt(index);  
        }  
        return index;  
    }  
}
```

## Ciclo 7

En este ciclo se busca comprobar si el método **breakLine()** puede procesar cadenas de texto con una longitud mayor.

### Código del test

```
@Test  
void lineExample7() {  
    assertEquals(  
        lineBreaker.breakText("test test test test", 9),  
        "test test\n\ttest test"  
    );  
}
```

### Mensaje de error obtenido al fallar el test

No hubo ningún fallo. Se pasó el test.

### Implementación que hace pasar el test

La implementación se corresponde con el código del ciclo anterior.

### Implementación después de refactorizar

La implementación se corresponde con el código del ciclo anterior.

## Ciclo 8

### Código del test

En este test se comprueba que a parte de introducirse bien un salto de línea, no se dejan espacios al principio de una línea.

```
@Test  
void lineExample8() {  
    assertEquals(  
        "test\n\ttest",  
        lineBreaker.breakText("test test", 4)|  
    );  
}
```

### Mensaje de error obtenido al fallar el test

Falla porque dichos espacios no fueron eliminados del principio de la siguiente línea.

```
J! org.opentest4j.AssertionFailedError: expected: <test>
test> but was: <test>
test>
    at es.urjc.code.daw.library.lineBreaker.TestLineBreaker.lineExample8(Test)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

## Implementación que hace pasar el test

Se introdujo una bifurcación para borrar el carácter espacio que está a continuación del salto de línea.

```
public class LineBreaker {
    public static final char NEXTLINE = '\n';

    public String breakText(String text, int lineLength) {
        if (text.length() <= lineLength) {return text;}
        StringBuilder textAux = new StringBuilder(text);
        int index = searchNearestSpace(textAux, lineLength);
        textAux.setCharAt(index, NEXTLINE);
        if (textAux.charAt(index + 1) == ' ') {
            textAux.deleteCharAt(index + 1);
        }
        return textAux.toString();
    }

    private int searchNearestSpace(
        StringBuilder text, int startingIndex
    ) {
        int index = startingIndex;
        char charAtIndex = text.charAt(index);
        while (index >= 0 && charAtIndex != ' ') {
            index--;
            charAtIndex = text.charAt(index);
        }
        return index;
    }
}
```

## Implementación después de refactorizar

Hicimos una implementación más general, que fuera capaz de borrar varios espacios al principio de la línea con el método privado **deleteSpaces()**.

```

package es.urjc.code.daw.library.lineBreaker;

public class LineBreaker {
    public static final char NEXTLINE = '\n';
    public static final char SPACE = ' ';

    public String breakText(String text, int lineLength) {
        if (text.length() <= lineLength) {return text;}
        StringBuilder textAux = new StringBuilder(text);
        int index = searchNearestSpace(textAux, lineLength);
        textAux.setCharAt(index, NEXTLINE);
        deleteSpaces(textAux, index + 1);
        return textAux.toString();
    }

    private int searchNearestSpace(
        StringBuilder text, int startingIndex
    ) {
        int index = startingIndex;
        char charAtIndex = text.charAt(index);
        while (index >= 0 && charAtIndex != ' ') {
            index--;
            charAtIndex = text.charAt(index);
        }
        return index;
    }

    private StringBuilder deleteSpaces(
        StringBuilder text, int startingIndex) {
        int index = startingIndex;
        char charAux = text.charAt(index);
        while (index < text.length() && charAux == SPACE) {
            text.deleteCharAt(index);
            charAux = text.charAt(index);
        }
        return text;
    }
}

```

## Ciclo 9

En este test se comprueba si **breakText()** procesa cadenas de texto que contengan varios espacios consecutivos.

### Código del test

```

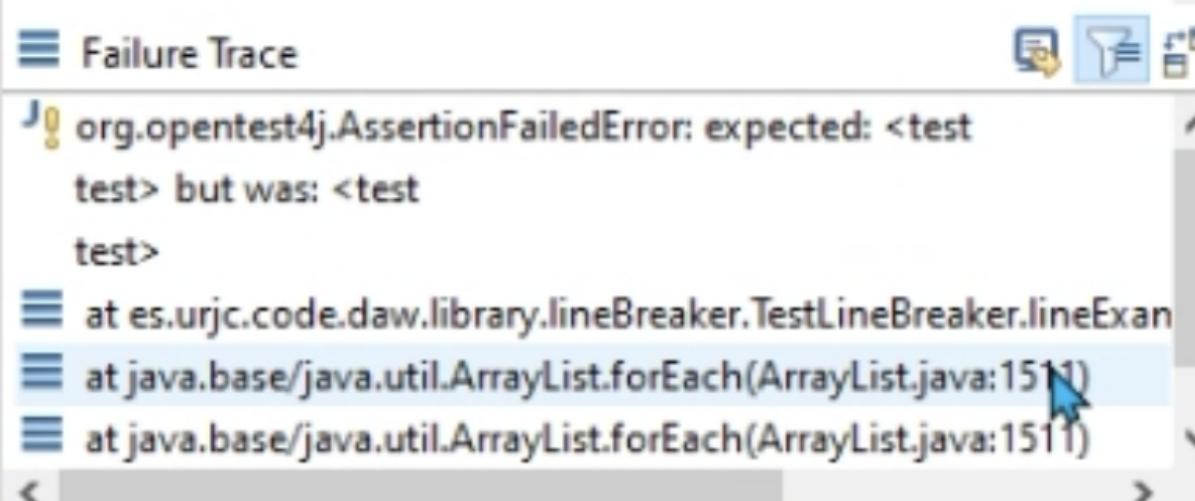
@Test
void lineExample9() {
    assertEquals("test\n\ttest", lineBreaker.breakText("test    test", 6));
}

```

### Mensaje de error obtenido al fallar el test

El error obtenido es debido a que cuando se sustituye el espacio por el salto de línea, la anterior sigue conteniendo un espacio al final. El método

`searchNearestSpace()` empieza a buscar espacios en el último carácter de la línea pero sólo da el índice del primer espacio que encuentre.



The screenshot shows a 'Failure Trace' window from a Java IDE. The title bar says 'Failure Trace'. The main area displays a stack trace starting with an assertion error:

```
J| org.opentest4j.AssertionFailedError: expected: <test
  test> but was: <test
  test>
  at es.urjc.code.daw.library.lineBreaker.TestLineBreaker.lineExan
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

The line 'at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)' is highlighted with a light blue background, and a mouse cursor is pointing at it. There are navigation arrows at the bottom left and right of the list.

## Implementación que hace pasar el test

```
public class LineBreaker {  
    public static final char NEXTLINE = '\n';  
    public static final char SPACE = ' ';  
  
    public String breakText(String text, int lineLength) {  
        if (text.length() <= lineLength) {return text;}  
        StringBuilder textAux = new StringBuilder(text);  
        int index = searchNearestSpace(textAux, lineLength);  
        textAux.setCharAt(index, NEXTLINE);  
        deleteSpaces(textAux, index + 1);  
        return textAux.toString();  
    }  
  
    private int searchNearestSpace(  
        StringBuilder text, int startingIndex  
    ) {  
        int index = startingIndex;  
        char charAtIndex = text.charAt(index);  
        while (index >= 0 && charAtIndex != ' ') {  
            index--;  
            charAtIndex = text.charAt(index);  
        }  
        return index; |  
    }  
  
    private StringBuilder deleteSpaces(  
        StringBuilder text, int index)  
    {  
        char charAux = text.charAt(index);  
        while (index < text.length() && charAux == SPACE) {  
            text.deleteCharAt(index);  
            charAux = text.charAt(index);  
        }  
        if (text.charAt(5) == ' ' && text.charAt(4) == ' ') {  
            text.deleteCharAt(4);  
            text.deleteCharAt(4);  
        }  
        return text;  
    }  
}
```

Para la implementación, como sabemos la localización de los espacios, introducimos en el método **deleteSpaces()** un if con las posiciones de los espacios del ejemplo.

## Implementación después de refactorizar

Para refactorizar, quitamos el if e introducimos una actualización del parámetro **startingIndex** dentro del bucle para eliminar los espacios de derecha a izquierda. Se dividió el método **deleteSpaces()** en dos métodos denominados: **deleteRightSpaces()** y **deleteLeftSpaces()** para eliminar los espacios al principio y al final de línea después de una división.

```
public static final char NEXTLINE = '\n';
public static final char SPACE = ' ';

public String breakText(String text, int lineLength) {
    text.trim();
    if (text.length() <= lineLength) {return text;}
    StringBuilder textAux = new StringBuilder(text);
    int index = searchNearestSpace(textAux, lineLength);
    textAux.setCharAt(index, NEXTLINE);
    deleteRightSpaces(textAux, index+1);
    deleteLeftSpaces(textAux, index-1);
    return textAux.toString();
}

private int searchNearestSpace(
    StringBuilder text, int startingIndex
) {
    int index = startingIndex;
    char charAtIndex = text.charAt(index);
    while (index >= 0 && charAtIndex != ' ') {
        index--;
        charAtIndex = text.charAt(index);
    }
    return index;
}

private void deleteRightSpaces(StringBuilder text, int startingIndex) {
    char charAux = text.charAt(startingIndex);
    while (startingIndex < text.length() && charAux == SPACE) {
        text.deleteCharAt(startingIndex);
        charAux = text.charAt(startingIndex);
    }
}
private void deleteLeftSpaces(StringBuilder text, int startingIndex) {
    char charAux = text.charAt(startingIndex);
    while (startingIndex >= 0 && charAux == SPACE) {
        text.deleteCharAt(startingIndex);
        startingIndex--;
        charAux = text.charAt(startingIndex);
    }
}
```

# Ciclo 10

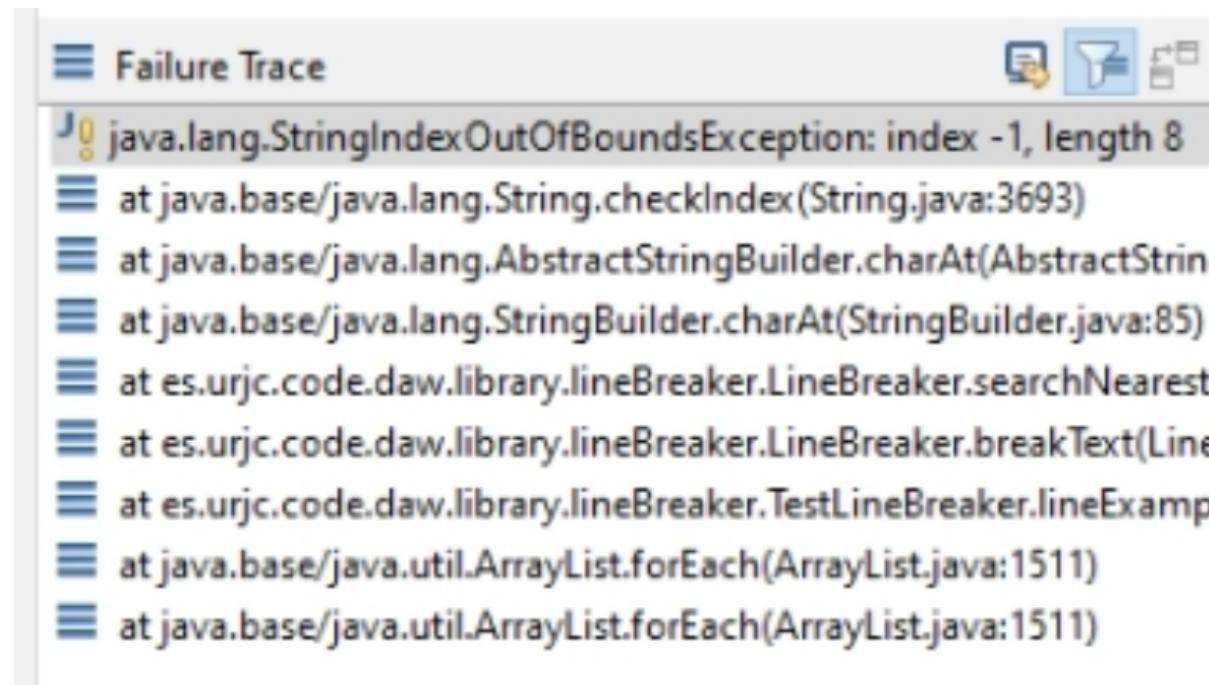
## Código del test

Se comprueba que si una línea no cabe en la longitud de línea dada esta debe ser dividida por un guión y un salto de línea.

```
@Test  
void lineExample10() {  
    assertEquals("test-\ntest", lineBreaker.breakText("testtest", 5));  
}
```

## Mensaje de error obtenido al fallar el test

Falla debido a que el método **searchNearestSpace()** no se espera que no exista ningún espacio en la línea y termina usando un índice no válido.



## Implementación que hace pasar el test

Introdujimos una ramificación en **breakText()** y en **searchNearestSpace()** para manejar el caso en el que no se encuentre un espacio en la línea.

```

public class LineBreaker {
    public static final char NEXTLINE = '\n';
    public static final char SPACE = ' ';
    public static final char HYPHEN = '-';

    public String breakText(String text, int lineLength) {
        if (text.equals("testtest")) return "test-\ntest";
        if (text.length() <= lineLength) {return text;}
        StringBuilder textAux = new StringBuilder(text);
        int index = searchNearestSpace(textAux, lineLength);
        textAux.setCharAt(index, NEXTLINE);
        deleteRightSpaces(textAux, index + 1);
        deleteLeftSpaces(textAux, index - 1);
        return textAux.toString();
    }

    private static int searchNearestSpace(StringBuilder text, int startingIndex) {
        int index = startingIndex;
        char charAtIndex = text.charAt(index);
        while (index > 0 && charAtIndex != SPACE && charAtIndex != NEXTLINE) {
            index--;
            charAtIndex = text.charAt(index);
        }
        if(charAtIndex == NEXTLINE)
            return -1;
        return index;
    }

    private void deleteRightSpaces(StringBuilder text, int startingIndex) {
        char charAux = text.charAt(startingIndex);
        while (startingIndex < text.length() && charAux == SPACE) {
            text.deleteCharAt(startingIndex);
            charAux = text.charAt(startingIndex);
        }
    }

    private void deleteLeftSpaces(StringBuilder text, int startingIndex) {
        char charAux = text.charAt(startingIndex);
        while (startingIndex < text.length() && charAux == SPACE) {
            text.deleteCharAt(startingIndex);
            startingIndex--;
            charAux = text.charAt(startingIndex);
        }
    }
}

```

## Implementación después de refactorizar

Arreglamos **searchNearestSpace()** para que pudiera detectar el final de una línea si se encuentra un salto de línea. También introdujimos una ramificación en **breakText()** para manejar el caso en el que no se encuentre un espacio en la línea. En el caso en el que no se encuentre un espacio, indica que el texto debe dividirse e introducir un guión al final de la línea junto con el salto a la siguiente.

```
public String breakText(String text, int lineLength) {  
    text.trim();  
    if (text.length() <= lineLength) {return text;}  
    StringBuilder textAux = new StringBuilder(text);  
    int index = searchNearestSpace(textAux, lineLength);  
    if(index > 0) {  
        textAux.setCharAt(index, NEXTLINE);  
        deleteRightSpaces(textAux, index+1);  
        deleteLeftSpaces(textAux, index-1);  
    }  
    else {  
        textAux.insert(lineLength-1, NEXTLINE);  
        textAux.insert(lineLength-1, HYPHEN);  
    }  
    return textAux.toString();  
}  
  
private int searchNearestSpace(  
    StringBuilder text, int startingIndex  
) {  
    int index = startingIndex;  
    char charAtIndex = text.charAt(index);  
    while (index > 0 && charAtIndex != SPACE && charAtIndex != NEXTLINE ) {  
        index--;  
        charAtIndex = text.charAt(index);  
    }  
    if(charAtIndex == NEXTLINE)  
        return -1;  
    return index;  
}
```

## Ciclo 11

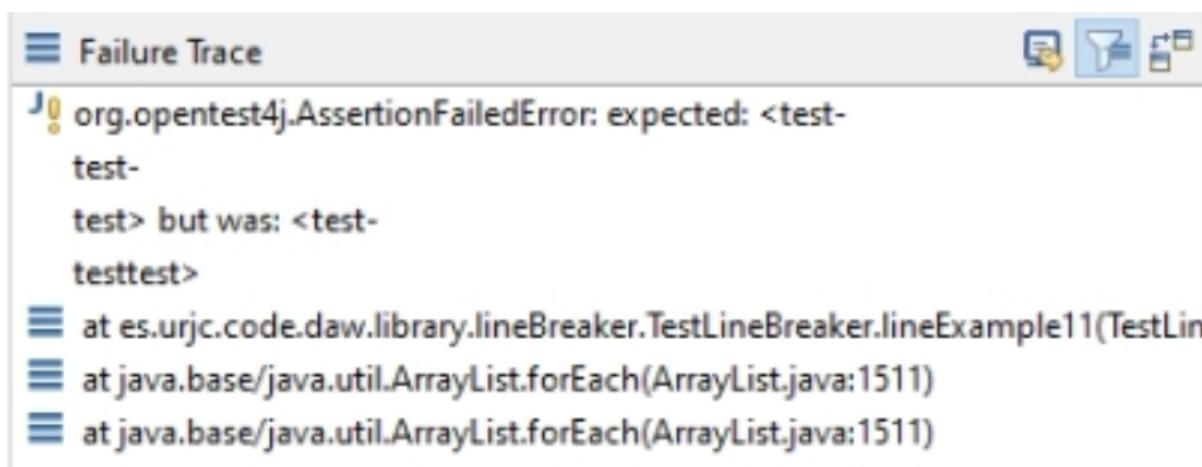
En este ciclo se comprueba que **breakText()** es capaz de dividir varias veces una cadena de texto que no contiene espacios.

## Código del test

```
@Test  
void lineExample11() {  
    assertEquals("test-\ntest-\ntest", lineBreaker.breakText("testtesttest", 5));  
}
```

## Mensaje de error obtenido al fallar el test

El error se debe a que nuestro código es capaz de hacer la división de la cadena una sola vez y el resto de la subcadena no es procesada.



## Implementación que hace pasar el test

```
public class LineBreaker {  
    public static final char NEXTLINE = '\n';  
    public static final char SPACE = ' ';  
    public static final char HYPHEN = '-';  
  
    public String breakText(String text, int lineLength) {  
        if(text.equals("testtesttest"))  
            return "test-\n test-\n test";  
        text.trim();  
        if (text.length() <= lineLength) {return text;}  
        StringBuilder textAux = new StringBuilder(text);  
        int index = searchNearestSpace(textAux, lineLength);  
        if(index > 0) {  
            textAux.setCharAt(index, NEXTLINE);  
            deleteRightSpaces(textAux, index+1);  
            deleteLeftSpaces(textAux, index-1);  
        }  
        else {  
            textAux.insert(lineLength-1, NEXTLINE);  
            textAux.insert(lineLength-1, HYPHEN);  
        }  
        return textAux.toString();  
    }  
  
    private int searchNearestSpace(  
        StringBuilder text, int startingIndex  
    ) {  
        int index = startingIndex;  
        char charAtIndex = text.charAt(index);  
        while (index > 0 && charAtIndex != ' ') {  
            index--;  
            charAtIndex = text.charAt(index);  
        }  
        return index;  
    }  
}
```

```
private void deleteRightSpaces(StringBuilder text, int startingIndex) {  
    char charAux = text.charAt(startingIndex);  
    while (startingIndex < text.length() && charAux == SPACE) {  
        text.deleteCharAt(startingIndex);  
        charAux = text.charAt(startingIndex);  
    }  
}  
private void deleteLeftSpaces(StringBuilder text, int startingIndex) {  
    char charAux = text.charAt(startingIndex);  
    while (startingIndex >= 0 && charAux == SPACE) {  
        text.deleteCharAt(startingIndex);  
        startingIndex--;  
        charAux = text.charAt(startingIndex);  
    }  
}
```

Para hacer pasar el test, hemos introducido un if al inicio del método con el ejemplo y devolvemos el resultado que se espera al procesarlo.

## Implementación después de refactorizar

En la refactorización hemos introducido un bucle que recorre la cadena para buscar las subcadenas y procesarlas de la misma manera en la que tratamos la subcadena del inicio. Se introduce la variable **lineDivider** que se corresponde con la posición en la que la cadena de texto se debe dividir. El bucle cesa cuando el valor de **lineDivider** es mayor que la longitud de la cadena de texto procesada.

```
public class LineBreaker {
    public static final char NEXTLINE = '\n';
    public static final char SPACE = ' ';
    public static final char HYPHEN = '-';

    public String breakText(String text, int lineLength) {
        int lineDivider = lineLength;
        text.trim();
        StringBuilder textAux = new StringBuilder(text);
        while(lineDivider < textAux.length()) {
            int index = searchNearestSpace(textAux, lineDivider);
            if(index > 0) {
                textAux.setCharAt(index, NEXTLINE);
                deleteRightSpaces(textAux, index+1);
                deleteLeftSpaces(textAux, index-1);
            }
            else {
                textAux.insert(lineDivider-1, NEXTLINE);
                textAux.insert(lineDivider-1, HYPHEN);
            }
            lineDivider = lineDivider + lineLength + 1;
        }
        return textAux.toString();
    }

    private int searchNearestSpace(
        StringBuilder text, int startingIndex
    ) {
        int index = startingIndex;
        char charAtIndex = text.charAt(index);
        while (index > 0 && charAtIndex != SPACE && charAtIndex != NEXTLINE) {
            index--;
            charAtIndex = text.charAt(index);
        }
        if(charAtIndex == NEXTLINE)
            return -1;
        return index;
    }

    private void deleteRightSpaces(StringBuilder text, int startingIndex) {
        char charAux = text.charAt(startingIndex);
        while (startingIndex < text.length() && charAux == SPACE) {
            text.deleteCharAt(startingIndex);
            charAux = text.charAt(startingIndex);
        }
    }

    private void deleteLeftSpaces(StringBuilder text, int startingIndex) {
        char charAux = text.charAt(startingIndex);
        while (startingIndex >= 0 && charAux == SPACE) {
            text.deleteCharAt(startingIndex);
            startingIndex--;
            charAux = text.charAt(startingIndex);
        }
    }
}
```

# Ciclo 12

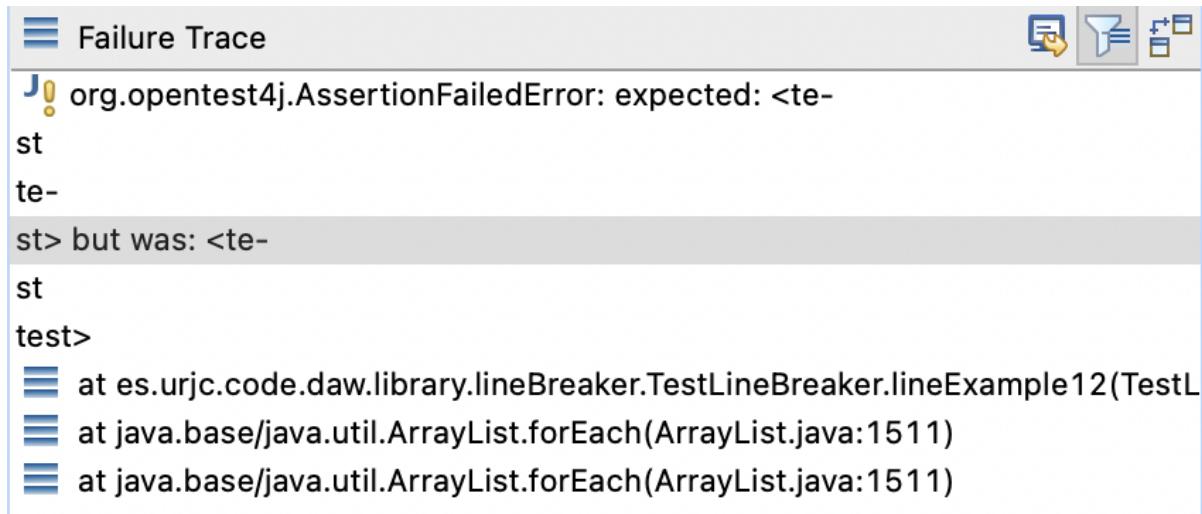
## Código del test

Se comprueba que se puede dividir una cadena con espacios, en líneas que no caben en la longitud de línea dada.

```
@Test  
void lineExample12() {  
    assertEquals("te-\nst\nte-\nst", lineBreaker.breakText("test test", 3));  
}
```

## Mensaje de error obtenido al fallar el test

Falla porque la tercera línea no se llegó a separar.



## Implementación que hace pasar el test

Primero se hizo pasar el test con una bifurcación al principio del método.

```

public class LineBreaker {
    public static final char NEXTLINE = '\n';
    public static final char SPACE = ' ';
    public static final char HYPHEN = '-';

    public String breakText(String text, int lineLength) {
        if (text.equals("test test") && lineLength == 3) {return "te-\nst\nte-\nst";}
        int lineDivider = lineLength;
        text.trim();
        StringBuilder textAux = new StringBuilder(text);
        while(lineDivider < textAux.length()) {
            int index = searchNearestSpace(textAux, lineDivider);
            if(index > 0) {
                textAux.setCharAt(index, NEXTLINE);
                deleteRightSpaces(textAux, index+1);
                deleteLeftSpaces(textAux, index-1);
            }
            else {
                textAux.insert(lineDivider-1,NEXTLINE);
                textAux.insert(lineDivider-1,HYPHEN);
            }
            lineDivider = lineDivider + lineLength + 1;
        }
        return textAux.toString();
    }

    private int searchNearestSpace(
        StringBuilder text, int startingIndex
    ) {
        int index = startingIndex;
        char charAtIndex = text.charAt(index);
        while (index > 0 && charAtIndex !=SPACE && charAtIndex !=NEXTLINE ) {
            index--;
            charAtIndex = text.charAt(index);
        }
        if(charAtIndex ==NEXTLINE)
            return -1;
        return index;
    }

    private void deleteRightSpaces(StringBuilder text, int startingIndex) {
        char charAux = text.charAt(startingIndex);
        while (startingIndex < text.length() && charAux == SPACE) {
            text.deleteCharAt(startingIndex);
            charAux = text.charAt(startingIndex);
        }
    }

    private void deleteLeftSpaces(StringBuilder text, int startingIndex) {
        char charAux = text.charAt(startingIndex);
        while (startingIndex >= 0 && charAux == SPACE) {
            text.deleteCharAt(startingIndex);
            startingIndex--;
            charAux = text.charAt(startingIndex);
        }
    }
}

```

## Implementación después de refactorizar

En la bifurcación después de **searchNearestSpace()**, en el caso en el que encontramos un espacio, actualizamos **lineDivider** para que sea igual a **index**. Esto se debe a que el lugar donde debe sustituirse el salto de línea es justo el índice donde se encontró el espacio.

```
public class LineBreaker {
    public static final char NEXTLINE = '\n';
    public static final char SPACE = ' ';
    public static final char HYPHEN = '-';

    public String breakText(String text, int lineLength) {
        int lineDivider = lineLength;
        text.trim();
        StringBuilder textAux = new StringBuilder(text);
        while (lineDivider < textAux.length()) {
            int index = searchNearestSpace(textAux, lineDivider);
            if (index > 0) {
                lineDivider = index;
                textAux.setCharAt(lineDivider, NEXTLINE);
                deleteRightSpaces(textAux, lineDivider + 1);
                deleteLeftSpaces(textAux, lineDivider - 1);
            } else {
                textAux.insert(lineDivider - 1, NEXTLINE);
                textAux.insert(lineDivider - 1, HYPHEN);
            }
            lineDivider = lineDivider + lineLength + 1;
        }
        return textAux.toString();
    }

    private int searchNearestSpace(
        StringBuilder text, int startingIndex
    ) {
        int index = startingIndex;
        char charAtIndex = text.charAt(index);
        while (index > 0 && charAtIndex != SPACE && charAtIndex != NEXTLINE) {
            index--;
            charAtIndex = text.charAt(index);
        }
        if (charAtIndex == NEXTLINE)
            return -1;
        return index;
    }

    private void deleteRightSpaces(StringBuilder text, int startingIndex) {
        char charAux = text.charAt(startingIndex);
        while (startingIndex < text.length() && charAux == SPACE) {
            text.deleteCharAt(startingIndex);
            charAux = text.charAt(startingIndex);
        }
    }

    private void deleteLeftSpaces(StringBuilder text, int startingIndex) {
        char charAux = text.charAt(startingIndex);
        while (startingIndex >= 0 && charAux == SPACE) {
            text.deleteCharAt(startingIndex);
            startingIndex--;
            charAux = text.charAt(startingIndex);
        }
    }
}
```

# Ciclo 13

## Código del test

Se comprueba que la división en varias líneas puede manejar diferentes tamaños de línea pero siempre menores al máximo dado.

```
@Test  
void lineExample13() {  
    assertEquals("test\n12345-\n67\ntest", lineBreaker.breakText("test 1234567 test", 6));  
}
```

## Mensaje de error obtenido al fallar el test

No hubo ningún fallo. Se pasó el test.

## Implementación que hace pasar el test

La implementación se corresponde con la del código anterior.

## Implementación después de refactorizar

La refactorización se corresponde con el código anterior.

# Ciclo 14

En este último ciclo, se comprueba que **breakText()** divide una cadena de entrada en varias subcadenas dada una longitud de línea pequeña.

## Código del test

```
@Test  
void lineExample14() {  
    assertEquals("12-\n34-\n56-\n789", lineBreaker.breakText("123456789", 3));  
}
```

## Mensaje de error obtenido al fallar el test

No hubo ningún fallo. Se pasó el test.

## Implementación que hace pasar el test

El código se corresponde con el ciclo anterior.

## Implementación después de refactorizar

Se refactorizó el código para que no saltara un **NullPointerException** al tomar el texto de entrada el valor **null**.

```
package es.urjc.code.daw.library.lineBreaker;

public class LineBreaker {
    public static final char NEXTLINE = '\n';
    public static final char SPACE = ' ';
    public static final char HYPHEN = '-';

    public static String breakText(String text, int lineLength) {
        if (text == null) {return null;}
        int lineDivider = lineLength;
        StringBuilder textAux = new StringBuilder(text.trim());
        while (lineDivider < textAux.length()) {
            int index = searchNearestSpace(textAux, lineDivider);
            if (index > 0) {
                lineDivider = index;
                textAux.setCharAt(lineDivider, NEXTLINE);
                deleteRightSpaces(textAux, lineDivider + 1);
                deleteLeftSpaces(textAux, lineDivider - 1);
            } else {
                textAux.insert(lineDivider - 1, NEXTLINE);
                textAux.insert(lineDivider - 1, HYPHEN);
            }
            lineDivider = lineDivider + lineLength + 1;
        }
        return textAux.toString();
    }

    private static int searchNearestSpace(StringBuilder text, int startingIndex) {
        int index = startingIndex;
        char charAtIndex = text.charAt(index);
        while (index > 0 && charAtIndex != SPACE && charAtIndex != NEXTLINE) {
            index--;
            charAtIndex = text.charAt(index);
        }
        if(charAtIndex == NEXTLINE)
            return -1;
        return index;
    }

    private static void deleteRightSpaces(StringBuilder text, int startingIndex) {
        char charAux = text.charAt(startingIndex);
        while (startingIndex < text.length() && charAux == SPACE) {
            text.deleteCharAt(startingIndex);
            charAux = text.charAt(startingIndex);
        }
    }

    private static void deleteLeftSpaces(StringBuilder text, int startingIndex) {
        char charAux = text.charAt(startingIndex);
        while (startingIndex >= 0 && charAux == SPACE) {
            text.deleteCharAt(startingIndex);
            startingIndex--;
            charAux = text.charAt(startingIndex);
        }
    }
}
```

## Test implementados:

Runs: 14/14      Errors: 0      Failures: 0

TestLineBreaker [Runner: JUnit 5] (0,056 s)

- lineExample10() (0,020 s)
- lineExample11() (0,003 s)
- lineExample12() (0,001 s)
- lineExample13() (0,001 s)
- lineExample14() (0,000 s)
- lineExample1() (0,001 s)
- lineExample2() (0,001 s)
- lineExample3() (0,001 s)
- lineExample4() (0,000 s)
- lineExample5() (0,001 s)
- lineExample6() (0,001 s)
- lineExample7() (0,001 s)
- lineExample8() (0,000 s)
- lineExample9() (0,000 s)

## Modificaciones en las clases BookService y DataBaseInitializer

```
public Book save(Book book) {  
    book.setDescription(LineBreaker.breakText(book.getDescription(), 10));  
    Book newBook = repository.save(book);  
    notificationService.notify("Book Event: book with title='"+newBook.getTitle()+" was created");  
    return newBook;  
}
```

En la clase de BookService se modificó el método **save()** aplicando a la descripción de un libro el método **breakText()** de LineBreaker antes de ser introducido a la base de datos.

```
@Component
public class DatabaseInitializer {

    @Autowired
    private BookService bookService;

    @PostConstruct
    public void init() {

        // Sample books

        bookService.save(new Book("SUEÑOS DE ACERO Y NEON",
            "Los personajes que protagonizan este relato sobreviven en una sociedad en c
        bookService.save(new Book("LA VIDA SECRETA DE LA MENTE",
            "La vida secreta de la mente es un viaje especular que recorre el cerebro y e
        bookService.save(new Book("CASI SIN QUERER",
            "El amor algunas veces es tan complicado como impredecible. Pero al final lo
        bookService.save(new Book("TERMINAMOS Y OTROS POEMAS SIN TERMINAR",
            "Recopilación de nuevos poemas, textos en prosa y pensamientos del autor. Ur
        bookService.save(new Book("LA LEGIÓN PERDIDA",
            "En el año 53 a. C. el cónsul Craso cruzó el Éufrates para conquistar Orient
    }
}
```

En cuanto a la inicialización de los libros de la base de datos, cambiamos el método `init()` usando el método `save()` de `BookService` en vez del método `save()` de `BookRepository` para que los libros de ejemplo se guardaran ya con la descripción formateada por `LineBreaker`.