

```

public class CountWords {
    public int count(String str) {
        int words = 0;
        char last = ' ';

        for (int i = 0; i < str.length(); i++) { // 1

            if (!isLetter(str.charAt(i)) && // 2
                (last == 's' || last == 'r')) {
                words++;
            }
            last = str.charAt(i); //3
        }
        if (last == 'r' || last == 's') {
            words;
        }
        return words;
    }
}

```

Pregunta: Explica qué hacen las líneas 1, 2, 3 en el código.

- 1: Recorre los caracteres de la cadena
- 2: Si el caracter no es una letra y el caracter anterior es la letra s o r.
- 3: actualizar el último caracter recorrido

```

@Test
void twoWordsEndingWithS() { // 1
    int words = new CountLetters().count("dogs cats");
    assertThat(words).isEqualTo(2);
}
@Test
void noWordsAtAll() { // 2
    int words = new CountLetters().count("dog cat");
    assertThat(words).isEqualTo(0);
}

```

Pregunta: Explica qué hacen las líneas 1, 2 del código. Presenta un informe generado por JaCoCo (www.jacoco.org/jacoco) o otra herramienta de código de tu preferencia en el ide del curso.

- 1: Prueba si dos palabras de la cadena terminan en s
- 2: Prueba si ninguna palabra de la cadena termina en s o r.

Coverage: CountWordsTest ×				
Element ▲	Class, %	Method, %	Line, %	Branch, %
▼ PruebasEstructurales	33% (1/3)	25% (1/4)	23% (10/42)	25% (10/40)
Clumps	0% (0/1)	0% (0/1)	0% (0/13)	0% (0/12)
CountWords	100% (1/1)	100% (1/1)	100% (10/10)	83% (10/12)
LeftPadUtils	0% (0/1)	0% (0/2)	0% (0/19)	0% (0/16)

```

@Test
void wordsThatEndInR() { // 1
    int words = new CountWords().count("car bar");
    assertThat(words).isEqualTo(2);
}

```

Pregunta: Explica la línea 1 y con el caso de prueba vuelve a ejecutar la herramienta de cobertura.

Explica los cambios obtenidos.

Coverage: CountWordsTest x				
Element ^	Class, %	Method, %	Line, %	Branch, %
▼ PruebasEstructurales	33% (1/3)	25% (1/4)	23% (10/42)	30% (12/40)
Clumps	0% (0/1)	0% (0/1)	0% (0/13)	0% (0/12)
CountWords	100% (1/1)	100% (1/1)	100% (10/10)	100% (12/12)
LeftPadUtils	0% (0/1)	0% (0/2)	0% (0/19)	0% (0/16)

El cambio es debajo de la columna Branch, en el caso anterior se obtenía 83% de cobertura y ahora 100%. Esto se debe a que la condición `last == 'r'` no se cumplía en ninguna prueba y con la última prueba agregada esta condición si se cumple.

```

public class LeftPadUtils {

    private static final String SPACE = " ";
    private static boolean isEmpty(final CharSequence cs) {
        return cs == null || cs.length() == 0;
    }

    /**
     * @param size
     * @param padStr
     * @return left
     * @code null}
     */
    public static String leftPad(final String str, final int size, String padStr) {
        if (str == null) { // 1
            return null;
        }
        if (isEmpty(padStr)) {
            padStr = SPACE; //2
        }

        final int padLen = padStr.length();
        final int strLen = str.length();
        final int pads = size - strLen;
        if (pads <= 0) { // 3
            return str;
        }

        if (pads == padLen) { // 4
            return padStr.concat(str);
        } else if (pads < padLen) { // 5
            return padStr.substring(0, pads).concat(str);
        } else { // 6
            final char[] padding = new char[pads];
            final char[] padChars = padStr.toCharArray();
            for (int i = 0; i < pads; i++) {
                padding[i] = padChars[i % padLen];
            }
            return new String(padding).concat(str);
        }
    }
}

```

Pregunta: Explica los comentarios 1, 2, 3, 4 y 5 del código anterior.

1: Si la cadena es null

2: Cadena para completar es espacio (" ")

3: Si no es necesario completar

4: Si el espacio disponible es el mismo que la longitud de la cadena para completar

5: Si el espacio disponible es menor que la longitud de la cadena para completar

```
public class LeftPadTest {
    @ParameterizedTest
    @MethodSource("generator")
    void test(String originalStr, int size, String padString,
              String expectedStr) { // 1
        assertThat(leftPad(originalStr, size, padString))
            .isEqualTo(expectedStr);
    }

    static Stream<Arguments> generator() { // 2
        return Stream.of(
            of(null, 10, "-", null), //T1
            of("", 5, "-", "-----"), //T2
            of("abc", -1, "-", "abc"), //T3
            of("abc", 5, null, " abc"), //T4
            of("abc", 5, "", " abc"), //T5
            of("abc", 5, "-", "--abc"), //T6
            of("abc", 3, "-", "abc"), //T7
            of("abc", 0, "-", "abc"), //T8
            of("abc", 2, "-", "abc") //T9
        );
    }
}
```

Pregunta: Explica las líneas 1,2 en el código anterior. Analiza el informe y responde qué sucede con las expresiones `if (pads == padLen)` y `else if (pads < padLen)`.

1: Define la prueba. Cada prueba depende de varios parámetros: cadena, tamaño final, cadena para completar y cadena esperada.

2: Método que contiene los valores que se usarán en las pruebas.

Coverage: LeftPadTest x			
Element ^	Class, %	Method, %	Line, %
PruebasEstructurales	33% (1/3)	50% (2/4)	41% (17/41)
Clumps	0% (0/1)	0% (0/1)	0% (0/13)
CountWords	0% (0/1)	0% (0/1)	0% (0/9)
LeftPadUtils	100% (1/1)	100% (2/2)	89% (17/19)

Se cubre el 89% de las líneas de código. Las condiciones `(pads == padLen)` y `(pads < padLen)` no se cumplen, por lo que el código del bloque `if` y `else if` correspondientes no se ejecutan.

```
static Stream<Arguments> generator() {
    return Stream.of(
        // ... otros aquí
        of("abc", 5, "--", "--abc"), // T10
        of("abc", 5, "---", "--abc"), // T11
        of("abc", 5, "-", "--abc") // T12
    );
}
```

Pregunta: Agrega estos tres casos de prueba adicionales a la prueba parametrizada, como se muestra en el listado y vuelve a ejecutar la herramienta de cobertura. Explica el informe obtenido, ¿es similar al anterior?. Explica tu respuesta.

Coverage: LeftPadTest x			
Element ^	Class, %	Method, %	Line, %
<ul style="list-style-type: none"> PruebasEstructurales <ul style="list-style-type: none"> Clumps CountWords LeftPadUtils 	33% (1/3)	50% (2/4)	46% (19/41)
Clumps	0% (0/1)	0% (0/1)	0% (0/13)
CountWords	0% (0/1)	0% (0/1)	0% (0/9)
LeftPadUtils	100% (1/1)	100% (2/2)	100% (19/19)

Ahora se cubre el 100% de las líneas de código ya que está el caso donde la cadena para completar tiene la misma longitud que la cantidad de caracteres faltantes y el caso donde la cadena para completar tiene mayor longitud que la cantidad de caracteres faltantes.

```
@Test
void sameInstance() {
    String str = "sometext";
    assertThat(leftPad(str, 5, "-")).isSameAs(str);
}
```

Pregunta: Agrega este caso de prueba adicional a la prueba parametrizada y vuelve a ejecutar la herramienta de cobertura. Explica el informe obtenido, ¿es similar al anterior?. Explica tu respuesta.

Coverage: LeftPadTest x			
Element ^	Class, %	Method, %	Line, %
<ul style="list-style-type: none"> PruebasEstructurales <ul style="list-style-type: none"> Clumps CountWords LeftPadUtils 	33% (1/3)	50% (2/4)	46% (19/41)
Clumps	0% (0/1)	0% (0/1)	0% (0/13)
CountWords	0% (0/1)	0% (0/1)	0% (0/9)
LeftPadUtils	100% (1/1)	100% (2/2)	100% (19/19)

El informe de cobertura es el mismo ya que la prueba agregada ejecuta líneas que ya se habían ejecutado en pruebas anteriores.

```
public class Clumps {
    /**
     * @param nums
     *
     * @return ...
     */
    public static int countClumps(int[] nums) {
        if (nums == null || nums.length == 0) { // 1
            return 0;
        }
    }
}
```

```

int count = 0;
int prev = nums[0];
boolean inClump = false;
for (int i = 1; i < nums.length; i++) {
    if (nums[i] == prev && !inClump) { // 2
        inClump = true;
        count += 1;
    }
    if (nums[i] != prev) { // 3
        prev = nums[i];
        inClump = false;
    }
}
return count;
}
}

```

Pregunta : Explica las líneas 1, 2 y 3 del código anterior.

- 1: Si el arreglo es nulo o vacío
- 2: Si el número es igual al anterior y la bandera inClump es falsa
- 3: Si el número no es igual al anterior

```

public class ClumpsOnlyStructuralTest {

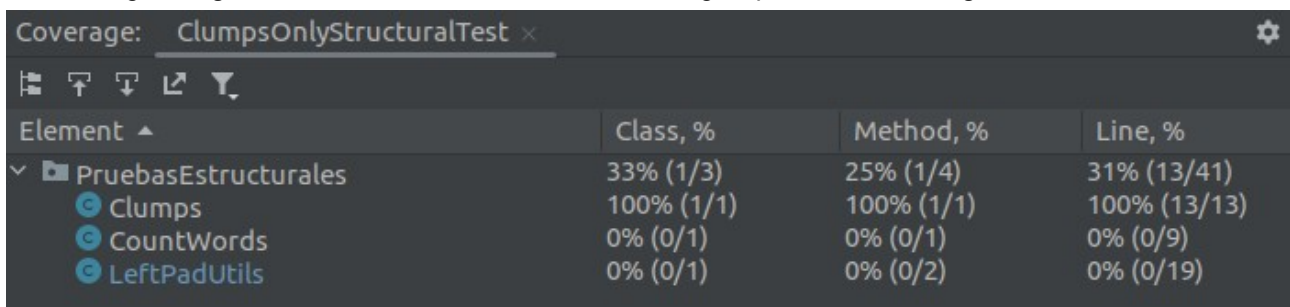
    @ParameterizedTest
    @MethodSource("generator")
    void testClumps(int[] nums, int expectedNoOfClumps) {
        assertThat(Clumps.countClumps(nums))
            .isEqualTo(expectedNoOfClumps);
    }

    static Stream<Arguments> generator() {
        return Stream.of(
            of(new int[] {}, 0), // vacío
            of(null, 0), // null
            of(new int[] {1,2,2,2,1}, 1), // 1 grupo
            of(new int[] {1}, 0), // 1 elemento

            // completa
            of(new int[] {2,2}, 1)
        );
    }
}

```

Pregunta: Escribe caso de prueba y vuelve a ejecutar la herramienta de cobertura. Explica el informe obtenido. ¿ Se logra una cobertura de ramas del 100%?. ¿Se puede confiar ciegamente en la cobertura? .



Coverage: ClumpsOnlyStructuralTest x				
Element ▲				
▼ PruebasEstructurales	33% (1/3)	25% (1/4)	31% (13/41)	
Clumps	100% (1/1)	100% (1/1)	100% (13/13)	
CountWords	0% (0/1)	0% (0/1)	0% (0/9)	
LeftPadUtils	0% (0/1)	0% (0/2)	0% (0/19)	

La cobertura de ramas obtenida es del 100%. No se puede confiar ciegamente en la cobertura de código ya que esta no garantiza que no existan errores en el código pero ayuda a detectar partes faltantes por probar.