

Ejercicio TDD

El desarrollo dirigido por pruebas (TDD o Test Driven Development) es una metodología de desarrollo de software mediante la cual se escriben todas las pruebas de un programa, con los resultados esperados, antes de crear el código propio del mismo. Este tipo de desarrollo supone una serie de ventajas en algunos aspectos frente a todo tipo de métodos de desarrollo y testing de programas. Algunas de estas ventajas podrían ser:

- Una mayor calidad del código, al promover un diseño más limpio y modular que puede ayudar a prevenir errores.
- Facilita el mantenimiento del código, puesto que al tener las pruebas realizadas desde el inicio, cualquier fallo posterior en el código se verá rápidamente en las pruebas. Además, la base de pruebas actuará como guía y documentación viva del programa.
- Incrementa la confianza al realizar cambios, puesto que al tener una base sólida de pruebas no será un problema cometer errores y podrán revertirse rápidamente, haciendo así que el desarrollo del código sea más efectivo y rápido.
- Fomenta el enfoque en los objetivos por etapas, dividiendo el código en diferentes fases no muy complejas desde un inicio.

El ciclo de TDD implica 3 pasos: Hacer una o varias versiones de una misma prueba que fallen; después se realiza el código mínimo necesario para que la prueba pase y, por último, se refactoriza el código para mejorar su calidad en caso de ser necesario.

Con todo esto, el TDD fomenta la creación de código más limpio, confiable y orientado a los requisitos, al mismo tiempo que acelera la detección temprana de errores. Es una práctica clave en la programación ágil y la entrega continua.

Ejercicio 1:

Lo primero que debe hacerse en este ejercicio es crear un proyecto de Java y en este crear un método dentro de una clase al que se quieran realizar tests.

You can also open a Java project folder, or create a new Java project by clicking the button below.

Create Java Project

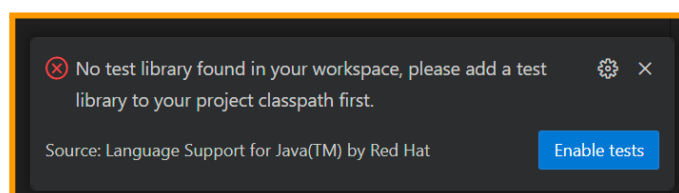
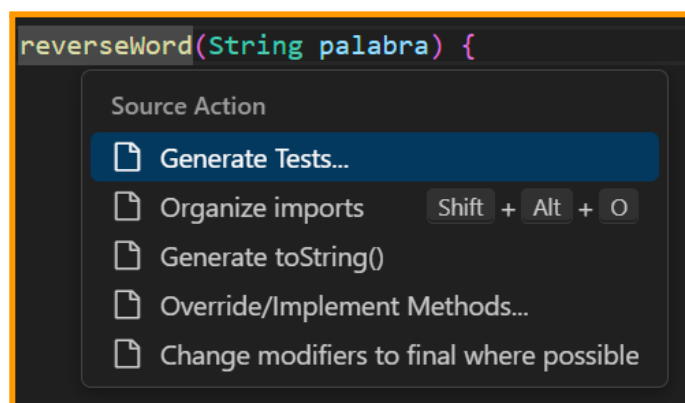
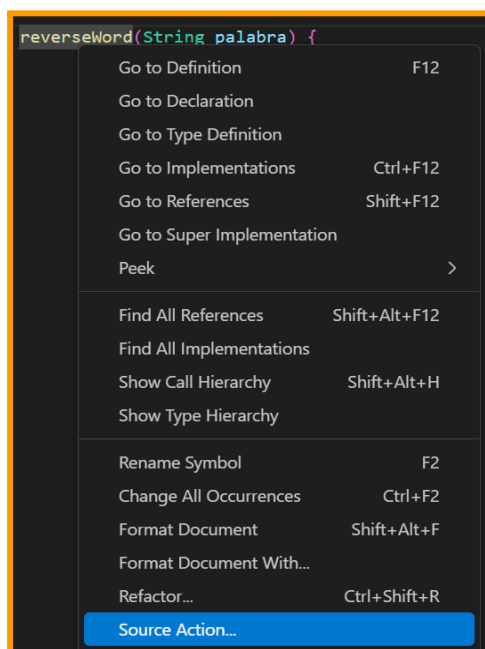
El método se llamará reverseWord y hará uso del string “palabra” para posteriormente darle la vuelta.

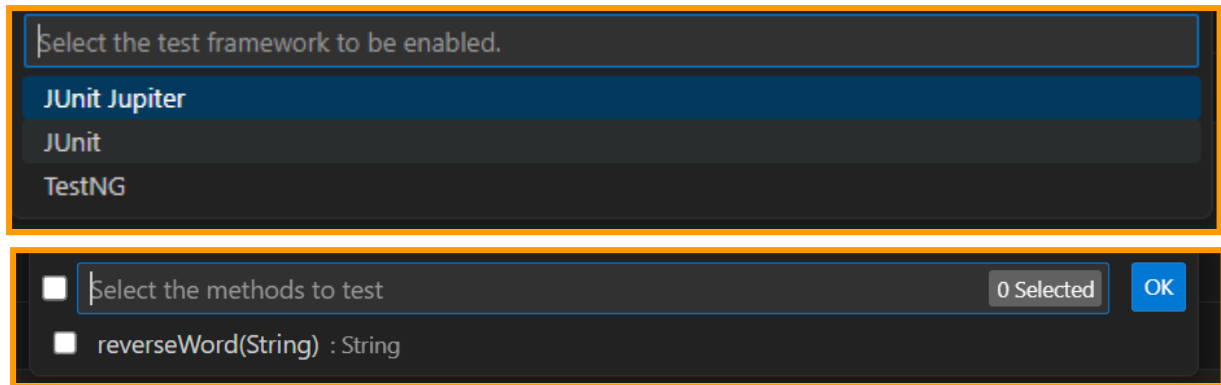
```
public class StringUtils {  
  
    public static String reverseWord(String palabra) {  
  
    }  
  
}
```

Una vez creado dicho método, se deja completamente vacío, añadiendo la cantidad de código mínimo como para que el código se pueda ejecutar y los tests se puedan realizar. Para ello se añade un return “” en este caso, de forma que no devuelva ningún texto y posiblemente no coincida el resultado del método con el resultado que va a ofrecer el test, puesto que esa es la primera fase a la hora de hacer un TDD, y se le conoce como firma.

```
public class StringUtils {  
    //crear la firma  
    public static String reverseWord(String palabra) {  
        return "";  
    }  
}
```

Una vez hecho esto, se crea el test del método, dando clic derecho sobre el nombre del mismo, seleccionando “source action”, “generate tests” y finalmente habilitando los tests de Junit y seleccionando el test que se va a crear dentro de la clase.





Tras esto, se elabora el código del test, dividiendo su contenido en 3 partes: arrange, donde se establece el string al que se le va a dar la vuelta, así como el resultado que debería dar de darse la vuelta, el resultado esperado.

```
public class StringUtilsTest {  
    @Test  
    public void testReverseWord() {  
        //arrange  
        String palabra="amigo";  
        String resultadoEsperado="ogima";
```

Después va el act, donde se llama al método original en base al valor establecido previamente.

```
//act  
String resultadoReal = StringUtils.reverseWord(palabra);
```

Y finalmente el assert, donde se comprueba este resultado con el esperado previo.

```
//assert  
Assert.assertEquals(resultadoEsperado, resultadoReal);
```

Al realizar el test de esta manera y con el método vacío, el test fallará, siendo este el objetivo; aun así, se repite el proceso al menos una vez con otros valores más para asegurarse de que falla exitosamente, ya que, en base al valor establecido en el método original, "", en caso de que en el test se le estuviese dando la vuelta a eso mismo, el test

```
4 public class StringUtilsTest {
5     @Test
6     public void testReverseWord_amigo() {
7         //arrange
8         String palabra="amigo";
9         String resultadoEsperado="ogima";
10
11         //act
12         String resultadoReal = StringUtils.reverseWord(palabra);
13
14         //assert
15         Assert.assertEquals(resultadoEsperado, resultadoReal); Expected [[ogima]] but was [[]]

```

Expected	Actual
-[ogima]	+[]

funcionaría. En este caso no sería tan necesario, pero en otros sí.

A la hora de realizar más tests, ya bien sea con la firma o con el test final, dichos tests se crean independientes del primero e identificables por el nombre respecto de los otros.

```
4 public class StringUtilsTest {
5     @Test
6     public void testReverseWord_amigo() {
7         //arrange
8         String palabra="amigo";
9         String resultadoEsperado="ogima";
10
11         //act
12         String resultadoReal = StringUtils.reverseWord(palabra);
13
14         //assert
15         Assert.assertEquals(resultadoEsperado, resultadoReal);
16     }
17     @Test
18     public void testReverseWord_perro() {
19         //arrange
20         String palabra="perro";
21         String resultadoEsperado="orrep";
22
23         //act
24         String resultadoReal = StringUtils.reverseWord(palabra);
25
26         //assert
27         Assert.assertEquals(resultadoEsperado, resultadoReal); Expected [[orrep]] but was [[]]

```

Expected	Actual
-[orrep]	+[]

```
29
30     @Test
31     public void testReverseWord_ornitorrinco() {
32         //arrange
33         String palabra="ornitorrinco";
34         String resultadoEsperado="ocnirrotinro";
35
36         //act
37         String resultadoReal = StringUtils.reverseWord(palabra);
38
39         //assert
40         Assert.assertEquals(resultadoEsperado, resultadoReal); Expected [[ocnirrotinro]] but was [[]]

```

Expected	Actual
-[ocnirrotinro]	+[]

Tras verificar esto, se procedería a realizar el código del método, ahora sí correctamente, para darle la vuelta a una palabra satisfactoriamente y nuevamente se volverían a hacer las pruebas para ver si funciona como se espera y el resultado que ha dado el test no ha sido mera coincidencia.

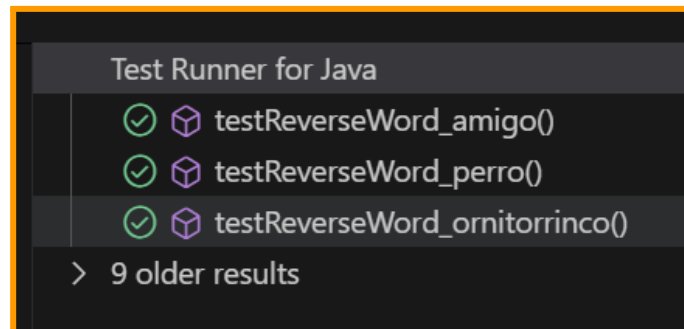
```
StringUtils.java > StringUtils
1 public class StringUtils {
2     public static String reverseWord(String palabra) {
3         String reverseString="";
4         for (int i=palabra.length()-1; i>=0; i--) {
5             reverseString = reverseString+palabra.charAt(i);
6         }
7         return reverseString;
8     }
9 }
```

En este código, el método reverseWord invierte una cadena al recorrerla de atrás hacia adelante utilizando un bucle for que comienza en el último índice (palabra.length() - 1) y termina en el primero. En cada iteración, se obtiene el carácter en la posición actual usando charAt(i) y se concatena al final de la cadena reverseString. Al finalizar el bucle, reverseString contiene la versión invertida de la cadena original, que se retorna como resultado.

Pruebas:

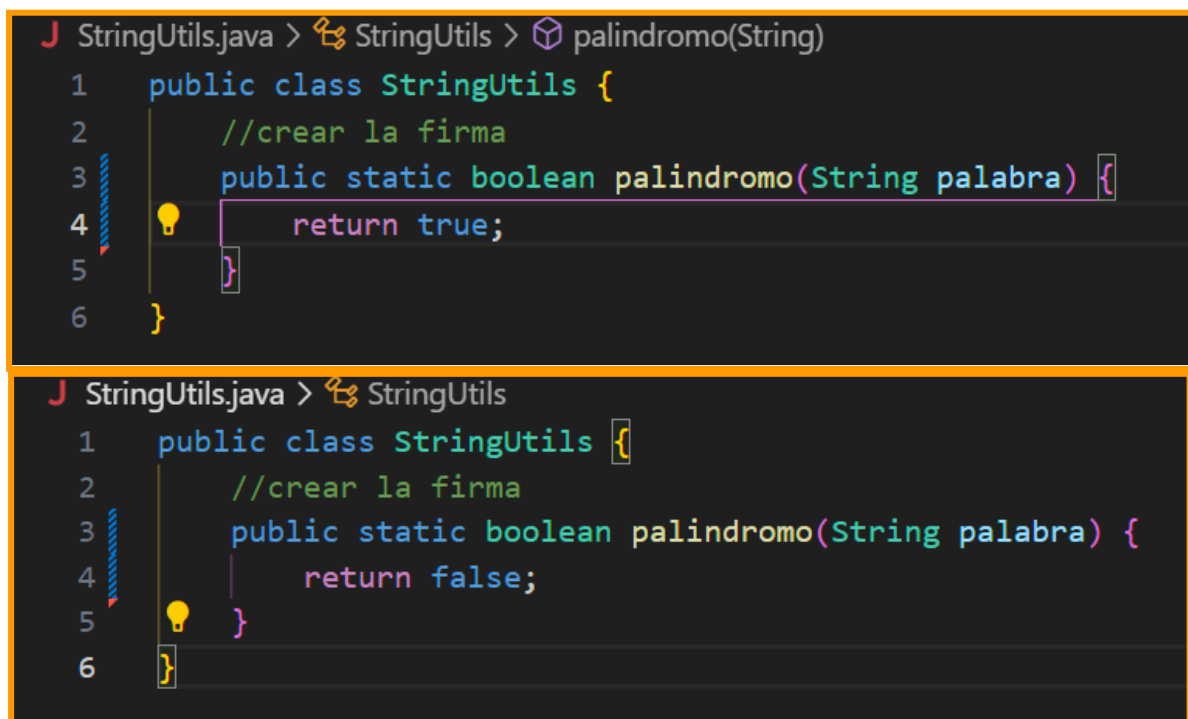
```
4 public class StringUtilsTest {
5     @Test
6     public void testReverseWord_amigo() {
7         //arrange
8         String palabra="amigo";
9         String resultadoEsperado="ogima";
10
11         //act
12         String resultadoReal = StringUtils.reverseWord(palabra);
13
14         //assert
15         Assert.assertEquals(resultadoEsperado, resultadoReal);
16     }
17     @Test
18     public void testReverseWord_perro() {
19         //arrange
20         String palabra="perro";
21         String resultadoEsperado="orrep";
22
23         //act
24         String resultadoReal = StringUtils.reverseWord(palabra);
25
26         //assert
27         Assert.assertEquals(resultadoEsperado, resultadoReal);
28     }
29     @Test
30     public void testReverseWord_ornitorrinco() {
31         //arrange
32         String palabra="ornitorrinco";
33         String resultadoEsperado="ocnirrotinro";
34
35         //act
36         String resultadoReal = StringUtils.reverseWord(palabra);
37
38         //assert
39         Assert.assertEquals(resultadoEsperado, resultadoReal);
40     }
41 }
```

Si pasa las pruebas, entonces el código estará completamente correcto.

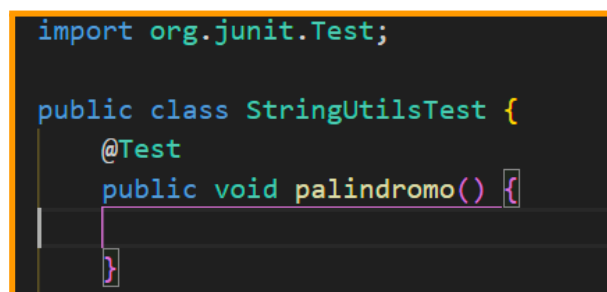


Ejercicio 2:

En este ejercicio, el objetivo es comprobar si una palabra es un palíndromo o no y devolver un valor booleano en función de ello. Para hacerlo, primero se crea el proyecto de Java, su clase y el método en cuestión, creando su firma que, en este caso, al devolver un boolean, solo podrá ser true o false.



Seguido del test del mismo, todo de la misma manera que se hizo en el ejercicio anterior.



Dentro del test se crean de nuevo las 3 etapas; en la primera, el arrange, se define la palabra que se va a usar de prueba para ver si es un palíndromo, y con la que posteriormente se ejecutará el método, así como el resultado esperado en caso de estar bien diseñado el código.

```
//arrange
String palabra="reconocer";
boolean resultadoEsperado=true;
```

En el Act, se establece cuál va a ser el resultado real, independiente del anterior, que será el que se ofrezca al llamar al método usando el valor previamente establecido.

```
//act
boolean resultadoReal = StringUtils.palindromo(palabra);
```

Por último, en el assert se comparan ambos resultados y se finaliza el test.

```
//assert
Assert.assertEquals(resultadoEsperado, resultadoReal);
```

Como valor de prueba, en este ejercicio se usará una palabra cualquiera, un string, y como resultados valores booleanos, true o false.

```
//arrange
String palabra="reconocer";
boolean resultadoEsperado=true;
```

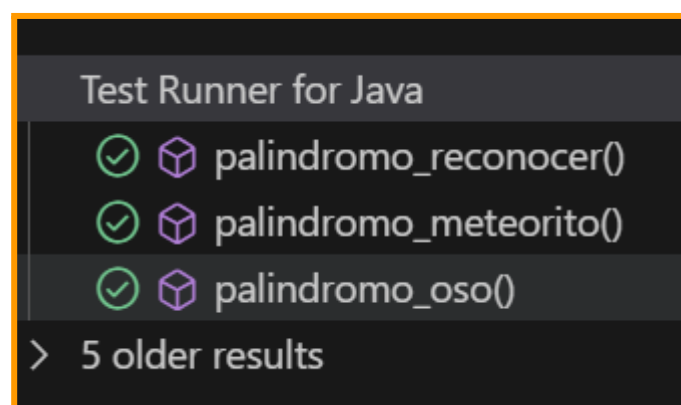
```
//arrange
String palabra="reconocer";
boolean resultadoEsperado=false;
```

De nuevo, se realizan más de 1 test, usando otros datos para los valores anteriores, ya que en este caso, en la firma, únicamente se puede establecer true o false, y los resultados serán iguales, de modo que es muy probable que de casualidad el test acierte incluso estando mal hecho.

Después, usando el mismo código para el test, se vuelven a realizar los tests, para verificar de manera definitiva si el código funciona.

```
4 public class StringUtilsTest {
5     @Test
6     public void palindromo_reconocer() {
7         //arrange
8         String palabra="reconocer";
9         boolean resultadoEsperado=true;
10
11         //act
12         boolean resultadoReal = StringUtils.palindromo(palabra);
13
14         //assert
15         Assert.assertEquals(resultadoEsperado, resultadoReal);
16     }
17     @Test
18     public void palindromo_meteorito() {
19         //arrange
20         String palabra="meteorito";
21         boolean resultadoEsperado=false;
22
23         //act
24         boolean resultadoReal = StringUtils.palindromo(palabra);
25
26         //assert
27         Assert.assertEquals(resultadoEsperado, resultadoReal);
28     }
29     @Test
30     public void palindromo_oso() {
31         //arrange
32         String palabra="oso";
33         boolean resultadoEsperado=true;
34
35         //act
36         boolean resultadoReal = StringUtils.palindromo(palabra);
37
38         //assert
39         Assert.assertEquals(resultadoEsperado, resultadoReal);
40     }
41 }
42
```

En caso de superar esos tests y descartarse que pueda ser coincidencia, el código estará correcto.



Ejercicio 3:

Tanto en este ejercicio como en el siguiente, el procedimiento a seguir va a ser el mismo a la hora de iniciar el ejercicio, crear la firma, el test, sus partes, el resto de tests a realizar, entre otras cosas a tener en cuenta. Por lo que la explicación en sí del ejercicio será mucho más corta y directa, para no repetir de nuevo todo lo anterior.

- Se crea la clase y la firma que contarán el número de consonantes de una palabra, haciendo que esta devuelva un número aleatorio, por ejemplo 1:

```
public class StringUtils {  
  
    public static int contarConsonantes(String palabra) {  
  
        return 1;  
    }  
}
```

- Se crea la estructura de los tests, estableciendo como resultado esperado un número que corresponda con el número real de consonantes que contiene la palabra a probar:

```
import org.junit.Assert;  
import org.junit.Test;  
  
public class StringUtilsTest {  
    @Test  
    public void testContarConsonantes() {  
        //arrange  
        String palabra="oso";  
        int resultadoEsperado=1;  
  
        //act  
        int resultadoReal = StringUtils.contarConsonantes(palabra);  
  
        //assert  
        Assert.assertEquals(resultadoEsperado, resultadoReal);  
    }  
}
```

- Se comprueba si falla el test:

```
1  import org.junit.Assert;
2  import org.junit.Test;
3
4  public class StringUtilsTest {
5      @Test
6      public void testContarConsonantes() {
7          //arrange
8          String palabra="oso";
9          int resultadoEsperado=1;
10
11         //act
12         int resultadoReal = StringUtils.contarConsonantes(palabra);
13
14         //assert
15         Assert.assertEquals(resultadoEsperado, resultadoReal);
16     }
17 }
18
```

- En este caso el test ha sido superado, a pesar de tener el código principal vacío; esto se debe a que la cantidad de consonantes que hay en la palabra oso es 1, igual al valor aleatorio que se ha puesto que devuelva el método.
- Se crean más tests, con diferentes palabras, para ver si el código puede fallar:

```
4  public class StringUtilsTest {
5      @Test
6      public void testContarConsonantes_oso() {
7          //arrange
8          String palabra="oso";
9          int resultadoEsperado=1;
10
11         //act
12         int resultadoReal = StringUtils.contarConsonantes(palabra);
13
14         //assert
15         Assert.assertEquals(resultadoEsperado, resultadoReal);
16     }
17
18     @Test
19     public void testContarConsonantes_judio() {
20         //arrange
21         String palabra="judio";
22         int resultadoEsperado=2;
23
24         //act
25         int resultadoReal = StringUtils.contarConsonantes(palabra);
26
27         //assert
28         Assert.assertEquals(resultadoEsperado, resultadoReal); Expected [2] but was [1]
29     }
30
31     @Test
32     public void testContarConsonantes_salchichon() {
33         //arrange
34         String palabra="salchichon";
35         int resultadoEsperado=7;
36
37         //act
38         int resultadoReal = StringUtils.contarConsonantes(palabra);
39
40         //assert
41         Assert.assertEquals(resultadoEsperado, resultadoReal); Expected [7] but was [1]
42     }
43 }
```

Expected	Actual
Expected [2] but was [1] testContarConsonantes_judio()	+1

- Una vez comprobado, se procede a realizar el código real del método original, que contará el número de consonantes en una palabra:

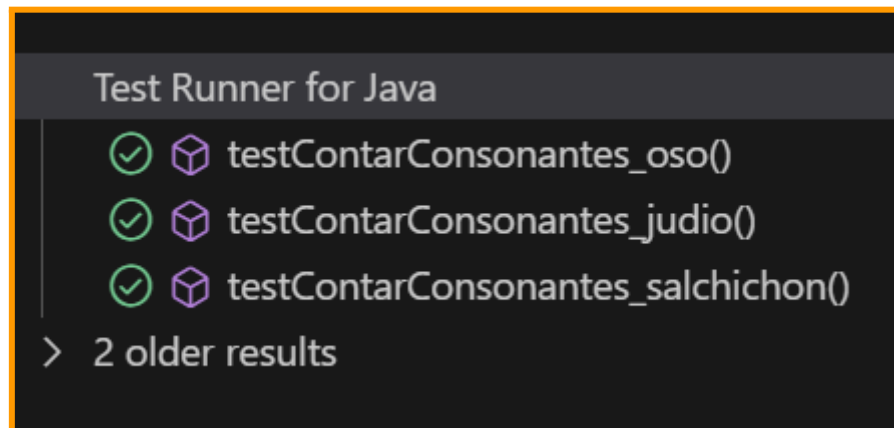
```
public class StringUtils {

    public static int contarConsonantes(String palabra) {
        int contador = 0;
        String vocales = "aeiouAEIOU";
        for (int i = 0; i < palabra.length(); i++) {
            char letra = palabra.charAt(i);
            if (Character.isLetter(letra) && !vocales.contains(String.valueOf(letra))) {
                contador++;
            }
        }
        return contador;
    }
}
```

- En este código, el método contarConsonantes cuenta las consonantes en una palabra. Recorre cada carácter de la palabra con un bucle for y lo convierte a minúscula usando Character.toLowerCase. Luego, verifica si el carácter es una letra entre 'a' y 'z' y no está en la cadena "aeiou", utilizando indexOf para comprobar si es vocal. Si cumple con las condiciones, incrementa un contador. Finalmente, devuelve el total de consonantes encontradas.
- Tras haber elaborado el código y habiendo pasado ya todos los pasos previos, se vuelven a realizar los mismos tests, para ver si ahora con el código correcto pueden ser superados.

```
4 public class StringUtilsTest {
5     @Test
6     public void testContarConsonantes_oso() {
7         //arrange
8         String palabra="oso";
9         int resultadoEsperado=1;
10
11         //act
12         int resultadoReal = StringUtils.contarConsonantes(palabra);
13
14         //assert
15         Assert.assertEquals(resultadoEsperado, resultadoReal);
16     }
17
18     @Test
19     public void testContarConsonantes_judio() {
20         //arrange
21         String palabra="judio";
22         int resultadoEsperado=2;
23
24         //act
25         int resultadoReal = StringUtils.contarConsonantes(palabra);
26
27         //assert
28         Assert.assertEquals(resultadoEsperado, resultadoReal);
29     }
30
31     @Test
32     public void testContarConsonantes_salchichon() {
33         //arrange
34         String palabra="salchichon";
35         int resultadoEsperado=7;
36
37         //act
38         int resultadoReal = StringUtils.contarConsonantes(palabra);
39
40         //assert
41         Assert.assertEquals(resultadoEsperado, resultadoReal);
42     }
43 }
```

- En caso de ser superados, el código estará correcto:



Ejercicio 4:

Mismo procedimiento que en el ejercicio anterior:

- Se crea la clase y la firma, que comprobarán si una palabra está formada únicamente por letras mayúsculas. Como valor a devolver, solo podrá devolver true o false al ser un valor booleano, por lo que se elige uno cualquiera, por ejemplo, true:

```
public class StringUtils {  
  
    public static boolean esSoloMayusculas(String palabra) {  
  
        return true;  
    }  
}
```

- Se crea la estructura de los test, estableciendo como resultado esperado true o false de nuevo, en base al String definido en la línea superior a este, que será con el que se realizará el test.

```
public class StringUtilsTest {  
    @Test  
    public void testEsSoloMayusculas() {  
        //arrange  
        String palabra="CARAMBANO";  
        boolean resultadoEsperado=true;  
  
        //act  
        boolean resultadoReal = StringUtils.esSoloMayusculas(palabra);  
  
        //assert  
        Assert.assertEquals(resultadoEsperado, resultadoReal);  
    }  
}
```

- Se comprueba si falla el test:

```
4 public class StringUtilsTest {
5     @Test
6     public void testEsSoloMayusculas() {
7         //arrange
8         String palabra="CARAMBANO";
9         boolean resultadoEsperado=true;
10
11         //act
12         boolean resultadoReal = StringUtils.esSoloMayusculas(palabra);
13
14         //assert
15         Assert.assertEquals(resultadoEsperado, resultadoReal);
16     }
17 }
```

- El test de nuevo no falla; esto se debe a lo mismo que en el ejercicio 2, y es que a la hora de hacer TDD con valores booleanos es muy probable que el test sea superado incluso si no debería hacerlo, ya que el código está vacío, puesto que los 2 únicos valores posibles son true o false.
- Se realizan más tests con diferentes palabras y no con todas las palabras en mayúsculas para ver si falla algún test.

```
4 public class StringUtilsTest {
5     @Test
6     public void testEsSoloMayusculas_CARAMBANO() {
7         //arrange
8         String palabra="CARAMBANO";
9         boolean resultadoEsperado=true;
10
11         //act
12         boolean resultadoReal = StringUtils.esSoloMayusculas(palabra);
13
14         //assert
15         Assert.assertEquals(resultadoEsperado, resultadoReal);
16     }
17     @Test
18     public void testEsSoloMayusculas_Payaso() {
19         //arrange
20         String palabra="Payaso";
21         boolean resultadoEsperado=false;
22
23         //act
24         boolean resultadoReal = StringUtils.esSoloMayusculas(palabra);
25
26         //assert
27         Assert.assertEquals(resultadoEsperado, resultadoReal); Expected [false] but was [true]
28     }
29     @Test
30     public void testEsSoloMayusculas_ANTIdoto() {
31         //arrange
32         String palabra="ANTIdoto";
33         boolean resultadoEsperado=false;
34
35         //act
36         boolean resultadoReal = StringUtils.esSoloMayusculas(palabra);
37
38         //assert
39         Assert.assertEquals(resultadoEsperado, resultadoReal); Expected [false] but was [true]
40     }
41 }
```

Expected	Actual
Expected [false] but was [true] testEsSoloMayusculas_Payaso()	
-false	+true

```
28 }
29 @Test
30 public void testEsSoloMayusculas_ANTIdoto() {
31     //arrange
32     String palabra="ANTIdoto";
33     boolean resultadoEsperado=false;
34
35     //act
36     boolean resultadoReal = StringUtils.esSoloMayusculas(palabra);
37
38     //assert
39     Assert.assertEquals(resultadoEsperado, resultadoReal); Expected [false] but was [true]
40 }
41 }
```

- Habiendo comprobado ya que los tests pueden fallar, se procede a realizar el código propio del método que contará correctamente el número de consonantes en una palabra:


```
public class StringUtils {  
  
    public static boolean esSoloMayusculas(String palabra) {  
        for (int i = 0; i < palabra.length(); i++) {  
            if (!Character.isUpperCase(palabra.charAt(i))) {  
                return false;  
            }  
        }  
        return true;  
    }  
}
```


- En este código, el método `esSoloMayusculas` verifica si una palabra contiene únicamente letras en mayúsculas. Recorre cada carácter de la palabra con un bucle `for` y utiliza `Character.isUpperCase` para comprobar si el carácter actual es mayúscula. Si encuentra un carácter que no lo sea, retorna `false` inmediatamente. Si el bucle completa el recorrido sin encontrar caracteres en minúsculas u otros que no sean mayúsculas, retorna `true`.
- Tras haber creado el código, se vuelven a hacer los mismos tests y se comprueba si son superados.


```
4 public class StringUtilsTest {  
5     @Test  
6     public void testEsSoloMayusculas_CARAMBANO() {  
7         //arrange  
8         String palabra="CARAMBANO";  
9         boolean resultadoEsperado=true;  
10  
11         //act  
12         boolean resultadoReal = StringUtils.esSoloMayusculas(palabra);  
13  
14         //assert  
15         Assert.assertEquals(resultadoEsperado, resultadoReal);  
16     }  
17     @Test  
18     public void testEsSoloMayusculas_Payaso() {  
19         //arrange  
20         String palabra="Payaso";  
21         boolean resultadoEsperado=false;  
22  
23         //act  
24         boolean resultadoReal = StringUtils.esSoloMayusculas(palabra);  
25  
26         //assert  
27         Assert.assertEquals(resultadoEsperado, resultadoReal);  
28     }  
29     @Test  
30     public void testEsSoloMayusculas_ANTIdoto() {  
31         //arrange  
32         String palabra="ANTIdoto";  
33         boolean resultadoEsperado=false;  
34  
35         //act  
36         boolean resultadoReal = StringUtils.esSoloMayusculas(palabra);  
37  
38         //assert  
39         Assert.assertEquals(resultadoEsperado, resultadoReal);  
40     }  
41 }  
42
```

- Si se superan, el código estará correcto.

Test Runner for Java

✓  testEsSoloMayusculas_CARAMBANO()

✓  testEsSoloMayusculas_Payaso()

✓  testEsSoloMayusculas_ANTIdoto()

> 3 older results