

## PREGUNTA 1: Implementacion.

Creamos la clase contar, que nos servira para saber cuantos grupos hay en un arreglo. Verificamos que cumpla con las precondiciones.

```
public class contarClumps {  
    /**  
     *Cuenta cuantos grupos hay  
     * @param array Longitud > 0 y array no nulo,  
     * @return Si Lon_arreglo = 0 o nula ,return 0  
     * Else cantidad de grupos.  
     */  
    public int contar(int[] array){  
        // verificamos las PreCondiciones  
        if(array.length==0 || array == null ){  
            return 0;  
        }  
        int grupos = 0;  
        int i = 0;  
        while( i < array.length){  
            int contador = 1;  
            while (i + 1 < array.length && array[i] == array[i + 1]) {  
                contador++;  
                i++;  
            }  
            i++;  
            if(contador >= 2){  
                grupos ++;  
            }  
            i++;  
        }  
        return grupos;  
    }  
}
```

## PREGUNTA 2:

100 % de cobertura

Class 	Class, %	Method, %	Line, %
contarClumps	100% (1/1)	100% (2/2)	100% (15/15)
main	0% (0/1)	0% (0/2)	0% (0/5)

Creamos algunas pruebas y vemos la cobertura.

- T1: que verifique correctamente cuantos grupos hay.

```
@Test
public void contarExitoso(){
    contarClumps cuenta = new contarClumps();
    int[] arreglo = { 1, 2, 2, 3, 4, 4, 4, 5, 5 };
    int numGrupos = cuenta.contar(arreglo);
    Assertions.assertEquals( expected: 3,numGrupos);
}
```

contarTest.contarExitoso x

Tests passed: 1 of 1 test – 92 ms

contarTest 92 ms

contarExitoso() 92 ms

Process finished with exit code 0

- T2: que no cumpla con la precondition.

```
@Test
public void violaPrecondicion(){
    contarClumps cuenta1 = new contarClumps();
    int[] arreglo = { 0 };
    int numGrupos = cuenta1.contar(arreglo);
    Assertions.assertEquals( expected: 3,numGrupos);
}
```

contarTest.violaPrecondicion x

Tests failed: 1 of 1 test – 45 ms

contarTest 45 ms

violaPrecondici 45 ms

org.opentest4j.AssertionFailedError:  
Expected :3  
Actual :0  
<Click to see difference>

- T3: cumple la precondition

```
@Test
public void cumplePrecondicion(){
    contarClumps cuenta1 = new contarClumps();
    int[] arreglo = {};
    int numGrupos = cuenta1.contar(arreglo);
    Assertions.assertEquals( expected: 0,numGrupos);
}
```

contarTest x

Tests passed: 2 of 2 tests – 126 ms

contarTest 126 ms

cumplePreco 124 ms

---- IntelliJ IDEA coverage runner ----

**Pregunta 3:** Ejecutando la herramienta de cobertura para mis 3 pruebas , notamos que en la rama contarTest, no llega al 100%, porque hay una prueba que falla.

Coverage: contarTest x				
Element	Class, %	Method, %	Line, %	Branch, %
contarClumps	100% (1/1)	100% (1/1)	100% (15/15)	91% (11/12)

Class	Class, %	Method, %	Branch, %	Line, %
contarClumps	100% (1/1)	100% (2/2)	91.7% (11/12)	100% (15/15)
main	100% (1/1)	50% (1/2)		66.7% (4/6)

**Pregunta 4:**

## **PREGUNTA 2:**

**1. Construye el SUT , completa y explica el código de prueba generado.**

```

1  @ExtendWith(MockitoExtension.class) //usando mockito
2  public class WordSelectionTest {
3      private WordSelection selection;
4
5      @Mock
6      private WordRepository repository;
7
8      @Test
9      public void reportsWordNotFound() throws WordRepositoryException {
10         selection = new WordSelection(repository);
11
12         doThrow(new WordRepositoryException("Exception")).when(repository).fetchWordByNumber(anyInt());
13
14         assertThatExceptionOfType(WordSelectionException.class)
15             .isThrownBy(() -> selection.getRandomWord());
16     }
17 }

```

Usaremos la extension de Mockito en esta prueba.

Usamos **@Mock** para crear el objeto simulado de la interfaz "wordRepository". usaremos doThrow de mockito para poder configurar el mock repository, para que lance la excepcion cuando llamamos al metodo fetchWordByNumber().

Usaremos aassertThat para verificar el caso en el que se intenta obtener una palabra del Repositorio, se espera que lance una excepcion cuando se quiera obtener una palabra.

**2. Indica los pasos Act y Assert de la prueba.**

ACT:

-al crear una instancia de de la clase WordSelection.

-El doThrow(new

WordRepositoryException("Exception")).when(repository).fetchWordByNumber(anyInt());

porque aqui configuramos el mock para que nos lance una excepcion cuando llamamos al metodo fetchWordByNumber con cualquier valor entero.

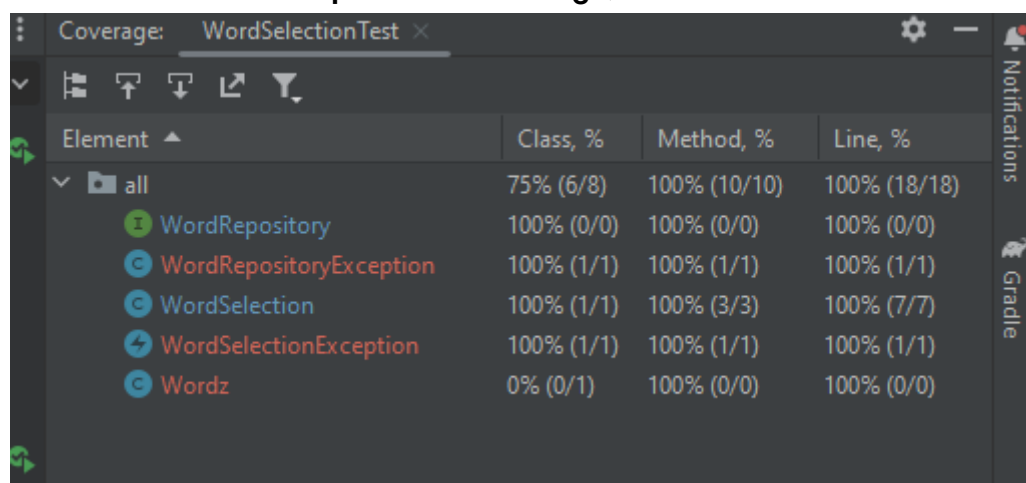
ASSERT:

La verificación seria cuando usamos el `assertThatExceptionOfType` para verificar que se lance una excepción de tipo `WordSelectionException` cuando llamamos al método `getRandomWord()`.

**3. Ejecuta la prueba. ¿Pasa la prueba?. Agrega la lógica necesaria para que la prueba pase.**

Si, agregado.

**4. Indica el código de cobertura obtenido y compara estos resultados con las actividades anteriores de la aplicación Wordz. ¿Qué diferencias encuentras?.**



Element	Class, %	Method, %	Line, %
all	75% (6/8)	100% (10/10)	100% (18/18)
WordRepository	100% (0/0)	100% (0/0)	100% (0/0)
WordRepositoryException	100% (1/1)	100% (1/1)	100% (1/1)
WordSelection	100% (1/1)	100% (3/3)	100% (7/7)
WordSelectionException	100% (1/1)	100% (1/1)	100% (1/1)
Wordz	0% (0/1)	100% (0/0)	100% (0/0)

Vemos que

**5. Muestra en que parte de tu código podemos usar dobles de prueba y la verificación de condiciones de error con TDD . Indica algunas decisiones de diseño en esta prueba sobre qué excepciones ocurren y dónde se usan.**

Al querer hacer una prueba para la clase `WordSelection` vemos que esta depende de la Interfaz `WordRepository`. Pero nosotros no queremos nada de esta clase.

Es por eso que simulamos esta clase usando mock.

En la prueba,usamos algunas excepciones:

Cuando configuramosel objeto simulado repository lanzamos una excepcion `WordRepositoryException`, para distinguir