Tabla de Contenidos

El concepto de JUnit Parameterized Test es un concepto relativamente nuevo y pertenece a JUnit 5. Cuando realizamos pruebas unitarias en muchas ocasiones nos encontramos con la necesidad de realizar el mismo test pasando diferentes grupos de parámetros. Vamos a ver un ejemplo sencillo de cómo esto funciona. El primer paso que deberemos hacer es construir un proyecto Maven y con él añadir las dependencias necesarias de Junit para crear JUnit Parameterized Test.

```
<dependency>
                        <groupId>org.junit.jupiter</groupId>
                        <artifactId>junit-jupiter-api</artifactId>
                        <version>5.9.0</version>
                        <scope>test</scope>
                </dependency>

                <!--
https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-par
ams -->
                <dependency>
                        <groupId>org.junit.jupiter</groupId>
                        <artifactId>junit-jupiter-params</artifactId>
                        <version>5.9.0</version>
                        <scope>test</scope>
                </dependency>
```

Una vez realizado este primer paso el siguiente paso es actualizar el proyecto de Maven para que por lo menos trabaje con Java 1.8.

```
	<properties>
		<maven.compiler.target>1.8</maven.compiler.target>
		<maven.compiler.source>1.8</maven.compiler.source>
	</properties>
```

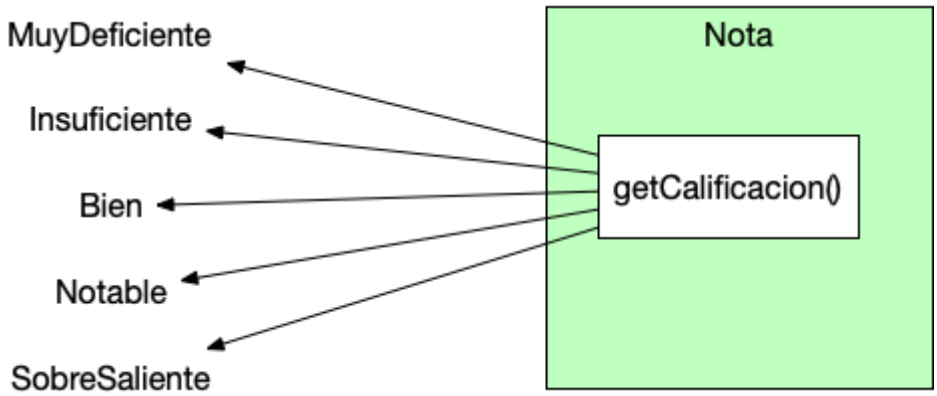
Hemos terminado de configurar el proyecto es momento de construir código y generar unas pruebas unitarias elementales. Para ello nos vamos a apoyar en la clase Nota . Esta clase se encarga de definir el concepto de nota y de devolvernos una calificación concreta dependiendo de la Nota que tengamos.

```
package com.arquitecturajava.models;

import java.util.Objects;

public class Nota {

	private String asignatura;
	private double valor;
	public String getAsignatura() {
		return asignatura;
	}
	public void setAsignatura(String asignatura) {
		this.asignatura = asignatura;
	}
	public double getValor() {
		return valor;
	}
	public void setValor(double valor) {
		if (valor<0 || valor>10) throw new
RuntimeException("valor de nota no valido");
		this.valor = valor;
	}
```

```
public Nota(String asignatura, double valor) {
        super();
        this.setAsignatura(asignatura);
        this.setValor(valor);
}
public String getCalificacion() {
        // sacame un mensaje cuando sea menor de cero
        // o mayor de 10
        if (valor>=0 && valor<=3) {
                return "MuyDeficiente";
        }else if (valor>3 && valor<5) {
                return "Insuficiente";
        }else if (valor>=5 && valor<6) {
                return "Suficiente";
        }else if (valor>=6 && valor<7) {
                return "Bien";
        }else if (valor>=7 && valor<9) {
                return "Notable";
        }else return "SobreSaliente";
}
@Override
public int hashCode() {
        return Objects.hash(asignatura, valor);
}
@Override
public boolean equals(Object obj) {
        if (this == obj)
                return true;
        if (obj == null)
                return false;
        if (getClass() != obj.getClass())
```

```
                        return false;
                Nota other = (Nota) obj;
                return Objects.equals(asignatura, other.asignatura)
                                && Double.doubleToLongBits(valor) ==
Double.doubleToLongBits(other.valor);
        }

}
```

La clave en este código la tenemos en el método getCalificacion().



Este método dependiendo del valor de la Nota nos devuelve Insuficiente, Notable etc.

```
public String getCalificacion() {
                // sacame un mensaje cuando sea menor de cero
                // o mayor de 10
                if (valor>=0 && valor<=3) {
                        return "MuyDeficiente";
                }else if (valor>3 && valor<5) {
                        return "Insuficiente";
                }else if (valor>=5 && valor<6) {
                        return "Suficiente";
                }else if (valor>=6 && valor<7) {
```

```
                return "Bien";
            }else if (valor>=7 && valor<9) {
                return "Notable";
            }else return "SobreSaliente";
        }
```

Usando JUnit Parameterized Test

Vamos a construir una prueba unitaria que nos permite Testear cuando uno nota es MuyDefiente . Es decir se encuentra ubicada entre 0 y 3.

```
package com.arquitecturajava.models;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

public class NotaTest2 {

        @Test
        public void muyDeficienteTestSencillo() {
                Nota nota = new Nota("matematicas",2);
                String calificacion= nota.getCalificacion();
                assertEquals("MuyDeficiente",calificacion);
        }

}
```

Este test es funcional pero en muchos casos nos interesará validar también los limites de la Nota. En este caso que los valores sean 0 o 3. Añadimos más código al test

```
@Test
        public void muyDeficienteTestSencillo() {
                Nota nota = new Nota("matematicas",2);
                String calificacion= nota.getCalificacion();
                assertEquals("MuyDeficiente",calificacion);
                Nota nota2 = new Nota("matematicas",2);
                String calificacion2= nota2.getCalificacion();
                assertEquals("MuyDeficiente",calificacion2);
                Nota nota3 = new Nota("matematicas",2);
                String calificacion3= nota3.getCalificacion();
                assertEquals("MuyDeficiente",calificacion3);
        }
```

Test y Limitaciones

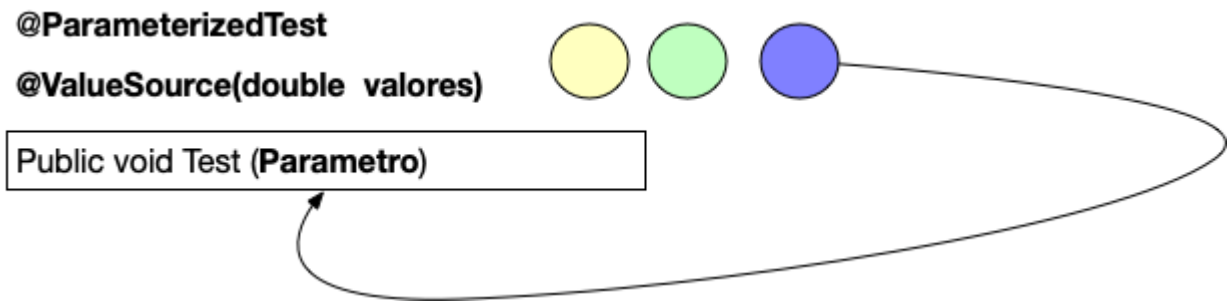
El Test nos es más práctico ya que valida los límites del valor . Sin embargo hay que escribir mucho más código para llegar a esa situación . En principio no nos puede parecer preocupante , pero cuando tengamos muchos Test será más código que leer. Podemos simplificar este código con un JUnit Parameterized Test que nos permite asociar a la prueba unitaria unos parámetros y pasárselos al método a través de una variable.

```
package com.arquitecturajava.models;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
```

```
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

public class NotaTest2 {
	@ParameterizedTest
	@DisplayName("Nota insuficiente valor:2 standar valor: 0,3
limites nota")
	@ValueSource (doubles= {2,0,3})
	public void muyDeficienteTest(double valorNota) {
		Nota nota2 = new Nota("matematicas",valorNota);
		String calificacion= nota2.getCalificacion();
		assertEquals("MuyDeficiente",calificacion);
	}


}
```

@ParameterizedTest

En este caso hemos usado la anotación @ParameterizedTest y @ValueSource para parametrizar el test y que reciba un parámetro con diversos valores.



Esto reduce el código de forma significativa y nos permite tener unas pruebas unitarias mucho más compactas.

Otros artículos relacionados

- JUnit Test Suite , TDD y organización
- @Before vs @BeforeClass en JUnit
- Spring Testing y el manejo de JUnit
- Test Parametrizados