JUnit 5
Este un framework utilizado para probar el código de producción. El cual fue desarrollado para poder probar el funcionamiento de las clases y métodos que componen nuestra aplicación, y asegurarnos que se comportan como deben ante distintas situaciones de entrada.

JUnit lo utilizamos cuando probamos un programa, lo ejecutamos con unos datos de entrada (casos de prueba) para verificar que el funcionamiento cumple los requisitos esperados.

Prueba Unitaria la definimos como la prueba de uno de los modelos que componen un programa.

JUnit 5 esta dividida en:
1. **JUnit Plaform** => es el entorno de ejecución donde las pruebas se correran
2. **JUnit Jupiter** => es donde se compone la API
3. **JUnit Vintage** => proporciona un motor de prueba para las versiones anteriores

Buenas Practicas para escribir código Limpio
1. Eres responsable por la calidad de tu código
2. Cuando escribas tu código usa nombres significativos(utilizar nombres que te diga la función que la **variable** hace)
3. Escribir código que exprese la intención (utilizar nombres en los **métodos** que expliquen claramente para que sirven)
4. Comenta tu código y explica su funcionalidad para que cuando otro desarrollador vea tu código lo entienda rápidamente
5. Cuando crees un método o una función, haz que este solo tenga una sola funcionalidad y la haga muy bien.
6. Hacer pruebas de integración y Unit Test

Buenas practicas para hacer pruebas unitarias
1. Probar solo una unidad de código a la vez: Cuando intentemos probar una unidad de código, esta unidad puede tener multiples cascos de uso. Siempre debemos probar cada caso de uso en un caso de prueba separado.
2. Hacer que cada prueba sea independiente de las demás
3. Simular todos los servicios externos
4. No hagas pruebas de configuración de unidad
5. Nombra tus pruebas de unidad de manera clara y consistente: Debes nombrar tus casos de prueba según lo que realmente hacen y probar
6. Escribir la pruebas para los métodos que tienen menos dependencia primero, y trabaje hacia arriba
7. Trate de que cada método de prueba de unidad realize exactamente una aserción.
8. Crear pruebas unitarias que apunten a excepciones.

9. Usar los métodos de Aserción apropiados.
10. Ponga los parámetros de aserción en el orden correcto.
11. Asegúrese de que el código de prueba este separado del código de producción
12. No Imprima nada en las pruebas unitarias
13. No confíe en las pruebas indirectas: No asuma que un caso de prueba particular también prueba otro escenario: En su lugar escriba otro caso de prueba para cada escenario.

1. **JUnit 5 Java Configuration pom.xml**

```
<!-- Junit 5 Dependencies for Java -->
<dependency>
	<groupId>org.junit.jupiter</groupId>
	<artifactId>junit-jupiter-engine</artifactId>
	<scope>test</scope>
</dependency>

<dependency>
	<groupId>org.junit.platform</groupId>
	<artifactId>junit-platform-runner</artifactId>
	<scope>{junit.platform.version}</scope>
</dependency>
```

2. Class to be Tested

```
package com.bazdigital.services;

public class FizzBuzz {

	public String play(int number) {

		if (number == 0)
			throw new IllegalArgumentException("Number must not be 0");

		if (number % 3 == 0)
			return "Fizz";

		if (number % 5 == 0)
			return "Buzz";

		return String.valueOf(number);
	}
}
```

3. **JUnit 5 Java Configuration JUnit5Class.java**

```
package com.bazdigital;
```

```
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import com.bazdigital.services.FizzBuzz;

public class Junit5 {


	private FizzBuzz fb;

	// use to initialize the object before each test
	@BeforeEach
	public void before() {
		fb = new FizzBuzz();
	}

	@DisplayName("Running a Test with JUnit 5")
	@Test
	public void test() {
		System.out.println("Test Succeded");
	}


	@DisplayName("Play FizzBuzz with number =  1")
	@Test
	public void testFizzBuzz() {
		String fizzBuzz = fb.play(1);
		Assertions.assertEquals(fizzBuzz, "1");
	}



	@DisplayName("Play FizzBuzz with number =  3")
	@Test
	public void testFizz() {
		String fizzBuzz = fb.play(3);
		Assertions.assertEquals(fizzBuzz, "Fizz");
	}
```

```
	@DisplayName("Play FizzBuzz with number =  5")
	@Test
	public void testBuzz() {
		String fizzBuzz = fb.play(5);
		Assertions.assertEquals(fizzBuzz, "Buzz");
	}


	// Test an exception
	@DisplayName("Don't play FizzBuzz with number =  0")
	@Test
	public void testZero() {
		Assertions.assertThrows(IllegalArgumentException.class,  () -> fb.play(0));
	};


	// reset the values of the object after each test
	@AfterEach
	public void tearDown() {
		fb = null;
	}

	// ignore a particular test
	@Disabled
	@DisplayName("Play FizzBuzz with number =  2")
	@Test
	public void testFizzBuzz2() {
		String fizzBuzz = fb.play(1);
		Assertions.assertEquals(fizzBuzz, "2");
	}

}
```