
imatia
innovation
We help you to do more

JUnit 5

- Cuando diseñamos una aplicación, es necesario probar nuestra implementación para comprobar que su funcionamiento es correcto.
- Es posible que, en un futuro, otro desarrollador modifique nuestro código para evolucionarlo, y exista algún comportamiento no deseado. Esto provocara un error que habrá que subsanar.
- Además, es posible que no tengamos en cuenta todas las posibilidades a la hora de diseñar un método, y se pueda da un error.

JUnit 5

- Para probar nuestro código, usamos herramientas que nos permitan asegurar que nuestro código sigue tratando los datos de la misma manera y obteniendo el mismo resultado
- Utilizaremos un framework para poder probar nuestro código llamado Junit, que se utiliza para realizar *pruebas unitarias*
- Una prueba unitaria nos permite examinar un fragmento de código y verificar que cumple con su cometido y afirmar su resultado

JUnit 5

- Junit5 permite utilizar todas las características de Java 8, como la programación funcional y métodos lambda.
- Para añadir la librería de JUnit 5 a nuestro proyecto, únicamente tenemos que importar la dependencia

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.8.2</version>
</dependency>
```

- Para poder crear test unitarios, tenemos que crear una clase en la misma ruta que tengamos nuestra clase original, pero bajo el pakage de *src/test/java*

  Clase:
  src/main/java/com/imatia/formaciones/lvl3_pruebas/junit/classes/Branch.java

  Test:
  src/test/java/com/imatia/formaciones/lvl3_pruebas/junit/classes/BranchTest.java

- Normalmente se nombran como la clase original más el sufijo *Test.* Branch -> Branch*Test*

- Los métodos que contendrán los test estarán creados dentro de una clase que no tendrá especificado ningún modificador de visibilidad, y los test serás los métodos marcados con la anotación *@Test*, no devolverán nada y tampoco tendrán ningún modificador de visibilidad explícito.

```
class Ejercicio65Test {

    @Test
    void testNewBranch() {
        Branch b = new Branch();
        assertNotNull(b.getEntityID());
        assertNotNull(b.getBranchID());
    }
}
```

JUnit 5
Assertions

- Los métodos de test utilizan los métodos de la clase *Assertions* para comprobar si el resultado que obtenemos de una operación es el resultado esperado, por ejemplo, si después de crear una instancia de un objeto, alguno de sus elementos no es nulo.
- Si un assertion falla, no se continúa con el resto de assertions del método.
- Podemos importar de manera estática la clase de *org.junit.jupiter.api.Assertions* para llamar a los métodos sin escribir constantemente *Assertions.assert….*

JUnit 5
Assertions

- Assertions más comunes:
  - assertEquals/assertNotEquals
  - assertNull/assertNotNull
  - assertSame/assertNotSame
  - assertThrows/assertNotThrows
  - assertTrue/assertFalse
  - assertAll
  - assumingThat
- **assertEquals/assertNotEquals**: Comprueba mediante el método *equals* de un objeto si ambos elementos son iguales.
- **assertNull/assertNotNull**: Comprueba si el elemento es nulo o no.

JUnit 5
Assertions

- **assertSame/assertNotSame** :
  Comprueba si ambos objectos son el mismo mediante el operador !=
- **assertTrue/assertFalse**:
  Comprueba si el elemento que se le pasa es verdadero o falso.
- **assertThrows/assertNotThrows**:
  Comprueba si que se lanza la excepción que se le indica o que no lanza ninguna excepción
- **assertAll**: Permite que se ejecuten todos los *assertion* que se le suministran, independientemente de si fallan o no.

JUnit 5
Assertions

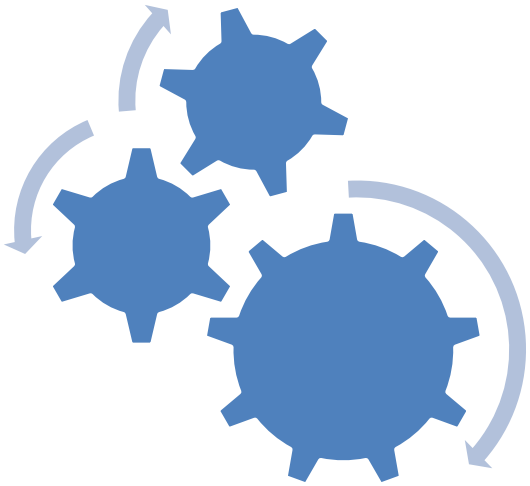
- **assumingThat:** Ejecuta la función lambda ejecutable (*que contendrá determinados assertions*) siempre y cuando la condición que recibe como parámetro de entrada sea verdadera. En caso que la condición sea falsa, no realiza la función lambda.

```
@Test
@DisplayName("Test if could add balance when minutes are even or odd")
void testAddBalanceEvenOdd(){
    boolean even = Calendar.getInstance().get(Calendar.MINUTE) % 2 == 0;
    assumingThat( even, () -> {
        a1.addBalance(new BigDecimal( val: "20.14"));
        a1.addBalance(new BigDecimal( val: "100"));
        assertEquals(this.a1.getBalance(), new BigDecimal( val: "120.14"));
    });

    assumingThat( !even, () -> {
        a1.addBalance(new BigDecimal( val: "30.14"));
        a1.addBalance(new BigDecimal( val: "100"));
        assertEquals(this.a1.getBalance(), new BigDecimal( val: "130.14"));
    });
}
```

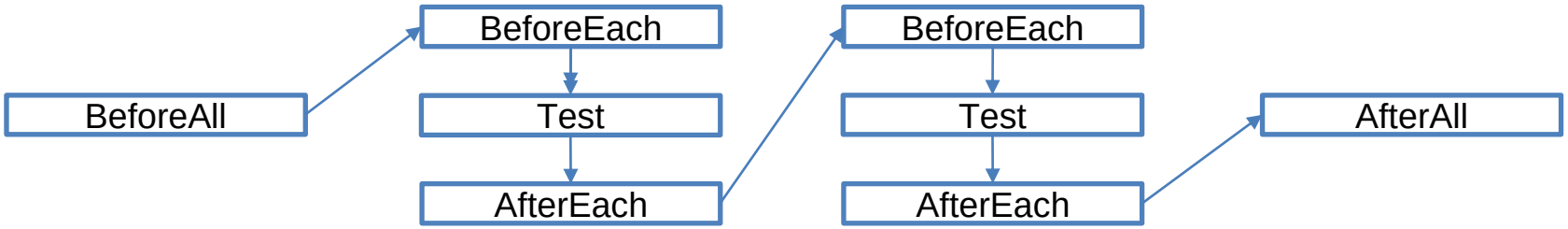
JUnit 5
Anotaciones

- Los métodos de test pueden contener otras anotaciones para modificar su visualización, organización o comportamiento, además de tener métodos auxiliares con anotaciones específicas para ellos.
- Todos los test deben probarse con instancias de objectos nuevas que no hayan sido previamente utilizadas en otros test.



JUnit 5
Anotaciones

- Existen 4 anotaciones especiales para todos los test de una clase
  - **BeforeAll**: Se ejecuta el método antes de todos los @Test y los @BeforeEach
  - **AfterAll**: Se ejecuta el método después de todos los @Test y los @AfterEach
  - **BeforeEach**: Se ejecuta antes de cada método @Test
  - **AfterEach**: Se ejecuta después de cada método @Test



JUnit 5
Anotaciones

- **@DisplayName**: Nos permite modificar el nombre que muestra el test en la pantalla.
- **@EnabledOnOs** / **@DisabledOnOS**: Indica el sistema operativo en el que se puede, o no, realizar el test
- **@EnabledOnJre** / **@DisabledOnJre**: Indica la versión de Java en la que se puede, o no, realizar el test
- **@EnabledIfSystemProperty** / **@DisabledIfSystemProperty**: Indica que el test se debe realizar, o no, si existe una propiedad del sistema con el valor que se le indica

JUnit 5
Anotaciones

- **@EnabledIfEnvironmentVariable / @DisabledIfEnvironmentVariable**: Indica que el test se debe realizar, o no, si existe una variable de entorno con el valor que se le indica
- **@RepeatedTest**: Permite realizar múltiples repeticiones del test. Necesita como mínimo que se indique el número de repeticiones (*value*). Además, se puede mostrar un mensaje específico para cada repetición (*name*) y se puede mostrar la repetición actual y total con {currentRepetition} y {totalRepetitions}. El método test también puede recibir por parámetro un *RepetitionInfo*, que contiene la información anterior

JUnit 5
Anotaciones

- **@ParameterizedTest**: Permite que el método test se repita y reciba diferentes parámetros a través de las siguientes anotaciones
  - **@ValueSource:** Una lista de elementos de un tipo determinado
  - **@CsvSource:** Un grupo de valores separados por una coma
  - **@CsvFileSource:** Lo mismo que la anterior, pero con un fichero externo
  - **@MethodSource:** El nombre del método que proveerá los datos
- Estos métodos se repiten por cada grupo de parámetros que reciban, y reciben como parámetros de entrada los valores que recibirán