

Drivers:

Requiere ejecutar una unidad de forma aislada para poder detectar defectos en el código de dicha unidad. Una unidad es un método Java.

Driver: conductor de la prueba. Contiene el código necesario para ejecutar el caso de prueba sobre el SUT

SUT: es el código que queremos probar. En este caso representa una unidad.

JUnit: Se puede utilizar para implementar drivers de pruebas unitarias y también de integración

- Prueba unitaria: Estamos interesados en probar una única unidad
- Prueba Integración: Estamos probando varias unidades (la SUT representa un conjunto de unidades).

Identificar Drivers con JUnit 5

```
package mismo.paquete.claseAprobar;  
  
import org.junit.jupiter.api.Test;  
  
class TrianguloTest {  
  
    @Test  
    void C01testQueNoHaceNada() {  
    }  
}
```

Los test deben de agruparse lógicamente con el SUT correspondiente (deben de pertenecer al mismo paquete)

La clase de pruebas tendrá el mismo nombre que la que contiene el SUT pero seguido de Test

El driver será un método void sin parámetros y está anotado con @Test. Cada método anotado con @Test implementará un driver para un único caso de prueba.

```
package psss;  
  
public class Triangulo {  
  
    public String tipo_triangulo  
        (int a, int b, int c) {  
        ...  
    }  
}
```

Driver

```
package psss;  
  
class TrianguloTest {  
    int a,b,c;  
    String real, esperado;  
  
    @Test  
    void testTipo_trianguloC1() {  
        a = 1;  
        b = 1;  
        c = 1;  
        resultadoEsperado = "Equilatero";  
        Triangulo tri = new Triangulo();  
        resultadoReal = tri.tipo_triangulo(a,b,c);  
        assertEquals(esperado, real);  
    }  
}
```

Sentencias Assert

JUnit proporciona sentencias (aserciones) para determinar el resultado de las pruebas y poder emitir el informe correspondiente. Son métodos estáticos que se utilizan para comparar el resultado esperado con el real. Si un assert falla salta AssertionFailedError.

```
import static  
org.junit.jupiter.api.Assertions.*;  
...  
assertEquals(...);
```

```
@Test  
void standardAssertions() {  
    /*todas las aserciones presentan  
    estas tres variantes: */  
    assertEquals(2, 2);  
    assertEquals(4, 4, "Mensaje opcional");  
    assertEquals(8, 8, () -> "Mensaje creado"  
                + "en tiempo de ejecución");  
}
```

Agrupación de Aserciones

Un test termina cuando se lanza la primera excepción no capturada en caso de que nuestros test contengan varias asserts usaremos siempre el método assertAll

```
// Añadimos un elemento a una colección llena
@Test
public void testC3Add() {
    int[] arrayEsperado = {1,2,3,4,5,6,7,8,9,10};
    int numElemEsperado = 10;

    int[] arrayEstadoInicial = Arrays.copyOf(arrayEsperado, arrayEsperado.length);
    DataArray colección = new DataArray(arrayEstadoInicial, 10);
    colección.add(11);
    //Agrupamos las aseraciones. SE EJECUTAN TODAS siempre
    assertAll("GrupoTestC3",
        ()> assertArrayEquals(arrayEsperado, colección.getColección()),
        ()> assertEquals(numElemEsperado, colección.size())
    ); //Se muestran todos los "fallos" producidos
}
```



Esta opción nos proporciona más información!!

Pruebas de Excepciones

- Si el resultado esperado de nuestra SUT es que lanza una excepción usaremos assertThrows.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertThrows;

@Test
void exceptionTesting() {
    //si sut() lanza la excepción de tipo ExpectedException
    //asignamos la excepción a la variable "exception"
    ExpectedException exception = assertThrows(ExpectedException.class,
        () -> sut(e1,e2));
    //mostramos el mensaje asociado a la excepción
    assertEquals("a message", exception.getMessage());
}
```

Si al ejecutar la SUT no se lanza la excepción el test fallará

- Si el resultado esperado del test es que no lance ninguna excepción usaremos assertDoesNotThrow

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertDoesNotThrow;

@Test
void NotExceptionTestingV1() {
    //si sut() lanza una excepción el test fallará
    assertDoesNotThrow(() -> sut(e1,e2), "Excepción lanzada");
}
```



Anotaciones de Test

- En el caso de que todos los test requieran las mismas acciones implementaremos dichas acciones comunes en un método anotado con @BeforeEach.
- @AfterEach en el caso de que sea necesario hacer algo después de cada test.
- En el caso de que las acciones previas o posteriores solo se tengan que hacer una vez.

```
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class OutputTest {
    static File output;

    @BeforeEach void createOutputFile() {
        output = new File(...);
    }
    @AfterEach void deleteOutputFile() {
        output.delete();
    }
    @BeforeAll static void initialState() {
        //initial code
    }
    @AfterAll static void finalState() {
        //final code
    }
    @Test void test1WithFile() {
        // code for test case objective
    }
    @Test void test2WithFile() {
        // code for test case objective
    }
}
```

Recordemos que no debemos asumir ningún orden en la ejecución de nuestros test, estos deben de ser independientes.

Etiquetado de los test

@Tag Esta anotación permite etiquetar a nuestros Test para filtrarlos, podemos usar varias etiquetas en un solo test.

```
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;

@Tag("fast")
@Tag("model")
class TaggingDemo {

    @Test
    @Tag("taxes") ←
    void testingTaxCalculation() {
        ...
    }
}
```

Test Parametrizados @ParameterizedTest y @ValueSource

- Si el código de nuestros Test es idéntico a excepción de los valores concretos del caso de prueba. Anotaremos con @ParameterizedTest y tendremos como parámetro los valores concretos.
- Si el test en concreto solo necesita un parámetro de tipo primitivo o String usaremos la anotación @ValueSource para indicar los valores para ese parámetro.

EJEMPLO de Test parametrizado usando @ValueSource

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

@ParameterizedTest
@ValueSource(strings = {"racecar", "radar", "able was I ere I saw elba"})
void palindromes(String candidate) {
    assertTrue(c.isPalindrome(candidate));
}
```

En este caso, los valores de la colección de parámetros son de tipo String

@ParameterizedTest y @MethodSource en el caso de que los test parametrizados requieran más de un parámetro, indicando el nombre del método en la etiqueta @MethodSource ("Nombre") devolviendo el método tuplas de valores correspondientes a la cantidad y tipo de parámetros de la SUT

EJEMPLO de Test parametrizado usando @MethodSource

```
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;

@ParameterizedTest(name = "User {1}, when Alert level is {2} should
                    have access to transporters of {0}")
@MethodSource("casosDePrueba")
void testParametrizado(boolean expected, Person user, Alert alertStatus) {
    transp.setAlertStatus(alertStatus);
    assertEquals(expected, transp.canAccessTransporter(user),
                () -> generateFailureMessage("transporter",
                                              expected, user, alertStatus));
}

//los valores devueltos por este método son los argumentos
//del método anotado con @ParameterizedTest
private static Stream<Arguments> casosDePrueba() {
    return Stream.of(
        Arguments.of(true, picard, Alert.NONE),
        Arguments.of(true, barclay, Alert.NONE),
        Arguments.of(false, lwaxana, Alert.NONE),
        Arguments.of(false, lwaxana, Alert.YELLOW),
        Arguments.of(false, q, Alert.YELLOW),
        Arguments.of(true, picard, Alert.RED),
        Arguments.of(false, q, Alert.RED)
    );
}

//método que construye y devuelve un mensaje de error en caso
//de que el resultado esperado no coincida con el real
private String generateFailureMessage(String system, boolean expected,
                                       Person user, Alert alertStatus) {
    String message = user.getFirstName() + " should";
    if (!expected) {
        message += " not";
    }
    message += " be able to access the " + system +
               " when alert status is " + alertStatus;
    return message;
}
```

Esto es simplemente un mensaje

El método casosDePrueba() devuelve un Stream con los Argumentos que se pasarán como parámetros al test parametrizado.
Cada objeto de tipo Arguments es un caso de prueba

Para poder Implementar y compilar los test:

Debemos incluir la librería dentro de nuestro POM

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.9.2</version>
  <scope>test</scope>
</dependency>
```

Da acceso

```
//código fuente de los tests en /src/test/java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Tag;
...
```

Si queremos implementar Test parametrizados:

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-params</artifactId>
  <version>5.9.2</version>
  <scope>test</scope>
</dependency>
```

```
//código fuente de los tests en /src/test/java
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;
import org.junit.jupiter.params.provider.ValueSource;
...
```

Para poder ejecutar los test lo haremos mediante la goal `surefire:test`, por lo que tendremos que indicar en el pom:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>3.0.0-M8</version>
</plugin>
```

mvn test

la goal `surefire:test` está asociada por defecto a la fase `test` de maven

Este ejecuta todos los métodos etiquetados con `@Test` o `@ParameterizedTest`

de las clases cuyos nombres acaben con `Test`, `Tests` o `TestCase.java`

Para filtrar los test excluidos podemos indicarlo en el pom mediante las propiedades `groups` y `excludedGroups`

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>3.0.0-M8</version>
  <configuration>
    <groups>etiqueta1,etiqueta2</groups>
    <excludedGroups>excluidos</excludedGroups>
  </configuration>
</plugin>
```

mvn test

En este ejemplo se ejecutarán los tests etiquetados como "etiqueta1" y también los etiquetados con "etiqueta2". También podemos excluir una (o varias) etiquetas del filtro

```
<properties>
  <filtrar.por>importantes,fase1</filtrar.por>
</properties>
...
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>3.0.0-M8</version>
  <configuration>
    <groups>${filtrar.por}</groups>
  </configuration>
</plugin>
```

mvn test

Se ejecutan los tests etiquetados como "importantes" más todos los tests etiquetados como "fase1"

mvn test -Dfiltrar.por=rapidos

Se ejecutan los tests etiquetados como "rapidos"

mvn test -Dfiltrar.por=""

Se ejecutan todos los tests

Ejercicios Práctica Drivers

⇒ ⇒ Ejercicio 1: *drivers para reservarButacas()*

Queremos automatizar la ejecución de los siguientes casos de prueba (Tabla A) asociados al método ***ppss.Cine.reservaButacas()***.

Tabla A.

	Datos de entrada		Resultado esperado	
	asientos[]	solicitados	(boolean + asientos[]) o ReservaException	
			boolean	asientos[]
C1	[]	3	ButacasException con mensaje "No se puede procesar la solicitud"	
C2	[]	0	false	[]
C3	[false, false, false, true, true]	2	true	[true, true, false, true, true]
C4	[true, true, true]	1	false	[true, true, true]

Los casos de prueba de la **Tabla A** se han obtenido aplicando el método del camino básico, por lo que dicha tabla es efectiva y eficiente. Deberías tener claro lo que eso significa.

Nota: Recuerda que podemos obtener conjuntos de prueba alternativos usando el mismo método, y que, si bien pueden tener una cardinalidad diferente, se consideran igualmente válidos (siempre y cuando no superen el valor de CC), y se obtengan a partir de un conjunto de caminos independientes.

```
3 public class Cine {  
4  
5  
6     public boolean[] asientos;  
7     public boolean reservaButacasV1(boolean[] asientos, int solicitados)  
8             throws ButacasException {  
9  
10        boolean reserva = false;  
11        int j = 0;  
12        int sitiosLibres = 0;  
13        int primerLibre;  
14  
15        if(solicitados > 0) {  
16  
17            if(solicitados > asientos.length){  
18                throw new ButacasException("No se puede procesar la solicitud");  
19            }  
20  
21            while ((j < asientos.length) && (sitiosLibres < solicitados)) {  
22                if (!asientos[j]) {  
23                    sitiosLibres++;  
24                } else {  
25                    sitiosLibres = 0;  
26                }  
27                j++;  
28            }  
29            if (sitiosLibres == solicitados) {  
30                primerLibre = (j - solicitados);  
31                reserva = true;  
32                for (int k = primerLibre; k < (primerLibre + solicitados); k++) {  
33                    asientos[k] = true;  
34                }  
35            }  
36  
37        }  
38  
39        this.asientos = asientos;  
40        return reserva;
```

a) Implementar los Drivers

Class CineTest 1

//Creamos un atributo público que setearemos antes de cada test
public Cine cine;

笪 Before Each

```
void setup () {
```

this.cine = new Cine();

1

// Empezamos con los test

@Test

void CineC1() {

boolean asientos [] = new Boolean [0];

Int solicitados = 3;

assert Equals ("No se puede procesar la solicitud", excepción.getMessage())

@Test

```
void CineCz () {
```

boolean asientos [] = new Boolean [0];

int solicitados = 0;

```
boolean reservaObtenida = assert.DoesNotThrow(() -> reserva.BestacasV1(asientos, solicitados));  
assert.All("Reserva Bestacas V1").
```

assert All ("ReservaButacas (2")

() → assert_array_equal (new Boolean [0], Cine.asientos)

() -> assertEquals (false , resultadoObtenido)) ; |

② Test

```
void CineC3 () {
    boolean asientos [] = { false, false, false, true, true };
    int solicitados = 2;
    boolean reservaObtenida = assertDoesNotThrow(() -> Cine.reservaButacasV1(asientos, solicitados));
    assertAll(() -> assertEquals(true, reservaObtenida),
              () -> assertEquals(new Boolean[] { true, true, false, true, true }, Cine.asientos));
}
```

③ Test

```
void CineC4 () {
    boolean asientos [] = { true, true, true };
    int solicitados = 1;
    boolean reservaObtenida = assertDoesNotThrow(() -> Cine.reservaButacasV1(asientos, solicitados));
    assertAll(() -> assertEquals(new Boolean[] { true, true, true }, Cine.asientos),
              () -> assertEquals(false, reservaObtenida));
}
```

Mismos Test Parametrizados

// Declaramos la función que contendrá los parámetros

```
private static Stream<Arguments> casosDePrueba ()
```

```
return Stream.of(
    Arguments.of( new Boolean[] {}, 3, new Boolean[] {}, false ),
    Arguments.of( new Boolean[] {}, 0, new Boolean[] {}, false ),
    Arguments.of( new Boolean[] { false, false, false, true, true }, 2, new Boolean[] { true, true, false, true, true },
                  false ),
    Arguments.of( new Boolean[] { true, true, true }, 1, new Boolean[] { true, true, true }, false ));
```

④ Parameterized Test

① Method Source ("casosDePrueba")

② Tag ("Parametrizado")

```
void CineParametrizado ( boolean [] asientos, int solicitados, boolean [] asientosEsperados, boolean reservaEsperada ) {
    boolean reservaObtenida = assertDoesNotThrow(() -> cine.reservaButacasV1(asientos, solicitados));
    assertAll(() -> assertEquals(asientosEsperados, cine.asientos),
              () -> assertEquals(reservaEsperada, reservaObtenida));
}
```

Configuraciones Maven:

• Correr test de CineTest sin la etiqueta "Parametrizado"

↳ clean test -D test=CineTest -DexcludedGroups=parametrizado

• Correr todos los test con la etiqueta Parametrizado:

↳ clean test -D Dgroups=parametrizado.