JUNIT

1. DESCRIPCIÓN

Las pruebas unitarias permiten probar la funcionalidad de los métodos de las clases.
JUnit es un framework (patrón o esquema que ayuda a la programación a estructurar el código y a ahorrar tiempo y esfuerzos a los programadores) popular para hacer pruebas unitarias en Java.
Este framework provee al usuario de herramientas, clases y métodos que le facilitan la tarea de realizar pruebas en su sistema y así asegurar su consistencia y funcionalidad.

1.1 MÉTODOS

A grandes rasgos, una clase de Test realizada para ser tratada por JUnit tiene una estructura con 4 tipos de métodos:
- Método setUp: Asignamos valores iniciales a variables antes de la ejecución de cada test. Si sólo queremos que se inicialicen al principio una vez, el método se debe llamar "setUpClass"
- Método tearDown: Es llamado después de cada test y puede servir para liberar recursos o similar. Igual que antes, si queremos que sólo se llame al final de la ejecución de todos los test, se debe llamar "tearDownClass"
- Métodos Test: Contienen las pruebas concretas que vamos a realizar.
- Métodos auxiliares.

Ejemplo de uso:

```
import org.junit.Assert;

import org.junit.AfterClass;
import org.junit.BeforeClass;

import org.junit.Test;


public class pruebaTest {


 @BeforeClass
```

```
public static void setUpClass() throws Exception {
 //Inicialización general de variables, escritura del log...
 }

 @AfterClass
 public static void tearDownClass() throws Exception {
 //Liberación de recursos, escritura en el log...
 }

 @Before
 public void setUp() {
 //Inicialización de variables antes de cada Test
 }

 @After
 public void tearDown() {
 //Tareas a realizar después de cada test
 }

 @Test
 public void comprobarAccion() {
 //Creamos el entorno necesario para la prueba
 //Usamos alguna de las funciones arriba descritas
 //para realizar la comprobación
 }

 public void funcionAuxiliar() {
 //tareas auxiliares
 }

}
```

1.2 EMPEZANDO CON JUNIT

Ejemplo:

Clase Ejemplo_junit.java

```
package ejemplo_junit;
```

```
public class Ejemplo_junit {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Calculadora obj = new Calculadora();
        int resultado=obj.sumarNumeros(3, 3);
        System.out.println(resultado);
    }

}
```

Clase Calculadora.java

```
package ejemplo_junit;


public class Calculadora {

    public int sumarNumeros(int a, int b){
        return a+b;
    }

}
```

Botón derecho sobre la clase → Tools → Create/Update Tests → Ok.

Esto genera las carpetas “Test Packages” y “Test Libraries”.

Dentro de “Test Packages” tenemos la clase “CalculadoraTest.java”, dentro del método testSumarNumeros() tenemos que comentar la línea de código fail("The test case is a prototype.");

Botón derecho sobre la clase → Test File → Open Test Results Window → Nos dirá si se completó o no con éxito.

Prueba 1: indicando en el método testSumarNumeros():

int a = 2;

int b = 4;

int expResult = 6;

Prueba 2: indicando en el método testSumarNumeros():

int a = 2;

```
int b = 4;
int expResult = 4;
```

Prueba 3: indicando en el método testSumarNumeros():

```
int a = 6;
int b = 4;
int expResult = 10;
assertEquals(10, result);
```

Prueba 4: indicando en el método testSumarNumeros():

```
int a = 6;
int b = 4;
int expResult = 15;
assertEquals(15, result);
```

1.3 ¿QUÉ SON LOS ASSERTS?

Una vez hemos creado las condiciones para probar que una funcionalidad concreta funciona es necesario que un validador nos diga si estamos obteniendo el resultado esperado o no. Para esta labor se definen una lista de funciones (incluidas en la clase Assert) que determinan el éxito o fracaso de una prueba.
Hay varios tipos de afirmaciones como booleanas, nulas, idénticas, etc.
Junit proporciona una clase llamada Assert, que proporciona un montón de métodos de afirmación útiles para escribir casos de prueba y para detectar fallos en las pruebas.
Los métodos assert son proporcionados por la clase **org.junit.Assert** que extiende la clase **java.lang.Object**.

Existen varios tipos de asserts (afirmaciones), que se resumen en:

AsserTrue / AssertFalse

static void assertTrue (boolean condition): afirma que la condición indicada es verdadera.

static void assertFalse (boolean condition): afirma que la condición indicada es falsa.

AssertNull / AssertNotNull

static void assertNull (Object actual): afirma que un objeto es null.

static void assertNotNull (Object actual): afirma que un objeto no es null.

AssertSame

static void assertSame (Object expected, Object actual): afirma que esperado y real se refieren al mismo objeto.

static void assertNotSame (Object unexpected, Object actual): afirma que esperado y real no se refieren al mismo objeto.

Ejemplo:

```
public class AssertSameTest {

  /*
   * Examples for each overloaded methods of assertSame
   */

  //public static void assertSame(Object expected, Object actual)
  @Test
  public void test1(){
   String str1="India";
   String str2="India";
   assertSame(str1,str2);
  }

  //public static void assertSame(Object expected, Object actual, String message)
  @Test
  public void test2(){

   String str1=new String("India");
   String str2=new String("India");
   assertSame(str1,str2,"str1 and str2 do not refer to same object");
  }

  //public static void assertSame(Object expected, Object actual, Supplier<String> messageSupplier)
  @Test
  public void test3(){
   String str1=new String("India");
   String str2=str1;
```

```
    assertSame(str1,str2,() -> "str1 and str2 do not refer to same object");
  }
}
```

Test1: Como str1 y str2 se refieren al mismo literal de cadena, este caso de prueba pasará.
Test2: Como str1 y str2 se refieren a dos nuevos objetos String diferentes, este caso de prueba fallará.
Test3: Como str1 se refiere al nuevo objeto y str2 se refiere al mismo objeto String (str2 = str1), este caso de prueba pasará.

AssertEquals

static void assertEquals (int expected, int actual): afirma que son iguales dos enteros.
static void assertEquals (int expected, int actual, String message): afirma que son iguales dos enteros con mensaje si no falla.
static void assertEquals (double expected, double actual, double delta, String message): afirma si son iguales dos doubles siendo, expected el primer tipo double que se va a comparar (es el tipo double que la prueba espera), actual es el segundo double que se va a comparar (es el tipo double producido por la prueba unitaria), delta es la precisión necesaria. Se producirá un error en la aserción solo si expected es diferente de actual en más de delta.

Assert Array Equals

static void assertArrayEquals (int[] expected, int [] actual): afirma si los arrays de enteros esperado y real son iguales.

fail (String message): hace que el método falle. Debe ser usado para probar que cierta parte del código no es alcanzable para que el test devuelva fallo hasta que se implemente el método de prueba. El parámero String es opcional.
Ejemplo:
@Test

```
public void test() {

    String str = null;

    if (str == null) {
```

```
            fail("El str proporcionado es null");

        }

    }
```

Hay más tipos que pueden consultarse en la API:

https://junit.org/junit5/docs/5.3.1/api/org/junit/platform/runner/JUnitPlatform.html

1.4 ANOTACIONES

Inicializador de clase de prueba: La anotación **@BeforeClass** marca un método como método de inicialización de clase de prueba. Un método de inicialización de clase de prueba se ejecuta solo una vez y antes que cualquiera de los otros métodos de la clase de prueba. Por ejemplo, en lugar de crear una conexión de base de datos en un inicializador de prueba y crear una nueva conexión antes de cada método de prueba, es posible que desee utilizar un inicializador de clase de prueba para abrir una conexión antes de ejecutar las pruebas. A continuación, puede cerrar la conexión con el finalizador de la clase de prueba.

Finalizador de la clase de prueba: La anotación **@AfterClass** marca un método como método finalizador de clase de prueba. Un método finalizador de clase de prueba se ejecuta solo una vez y después de que todos los demás métodos de la clase de prueba hayan finalizado.

Prueba de inicializador: La anotación **@Before** marca un método como método de inicialización de prueba. Se ejecuta un método de inicialización de prueba antes de cada caso de prueba en la clase de prueba. No se requiere un método de inicialización de prueba para ejecutar pruebas, pero si necesita

inicializar algunas variables antes de ejecutar una prueba, utilice un método de inicialización de prueba.

Finalizador de prueba: La anotación **@After** marca un método como método finalizador de prueba. Se ejecuta un método de finalizador de prueba después de cada caso de prueba en la clase de prueba. No se requiere un método de finalizador de prueba para ejecutar pruebas, pero es posible que necesite un finalizador para limpiar los datos que se requirieron al ejecutar los casos de prueba.

@Test: Identifica un método como método test. Admite los parámetros:

a) Parámeto **expected:** Se utiliza para indicar que un método lance una determinada excepción; falla cuando no se lanza la excepción o no se lanza la indicada.

b) Parámetro **timeout:** Se utiliza para que una prueba falle si un método tarda en ejecutarse más tiempo de lo normal. El tiempo se indica en milisegundos.

@Ignore: ignora el método de test. Es útil cuando el código a probar ha cambiado y el caso de uso no ha sido todavía adaptado.

Test parametrizados

Es una forma cómoda de poder ejecutar pruebas unitarias en las que lo que queremos es comprobar que para varios valores obtenemos varios resultados.

En este tipo de pruebas podemos conseguir que JUnit repita rápidamente una o varias pruebas unitarias mediante el uso de casos de prueba proporcionados como conjunto.

@RunWith: sirve para indicarle a Junit que lance la prueba unitaria correspondiente con un runner particular.

Se necesita para ello un método especial que tiene que ser public static, que devuelva un objeto de tipo Iterable (ArrayList, Colección...etc) y que además sea array Object. Este método contendrá la anotación **@Parameters.**

Cuando se ejecute la prueba unitaria de esta clase, Parameterized buscará el método que contenga esta anotación y que sea compatible con las características del método que hemos descrito anteriormente. Los arrays de objetos tendrán que tener tantas posiciones como parámetros tenga el constructor de la clase.

Suites

Cómo podemos lanzar un grupo de pruebas de forma más o menos controlada y a ser posible en orden, es decir, como se pueden lanzar varias pruebas unitarias una detrás de otra por medio de los suites, que nos dejan agrupar clases con pruebas similares.

Se utiliza la anotación **@Suite.SuiteClasses** para indicar las clases a lanzar y el orden de las mismas. En la anotación **@RunWith** se tendrá que indicar Suite.class.

Para crear una clase de este tipo en NetBeans: botón derecho sobre el paquete→ new → other → Unit Tests → Test Suite