Módulo: Entornos de Desarrollo

Unidad 5: Diseño y realización de Pruebas

Pruebas con Junit4 en NetBeans

- En este ejercicio se va a desarrollar un ejemplo completo de pruebas unitarias en Junit 4 para Netbeans en Java. Este desarrollo completo se puede encontrar en la web oficial del NetBeans:

  https://netbeans.apache.org/kb/docs/java/junit-intro.html

Objetivos:

- Realizar pruebas unitarias de clases y funciones.
- Implementar pruebas automáticas.

Recursos:

- Acceso a Internet.
- Procesador de texto

Resolución:

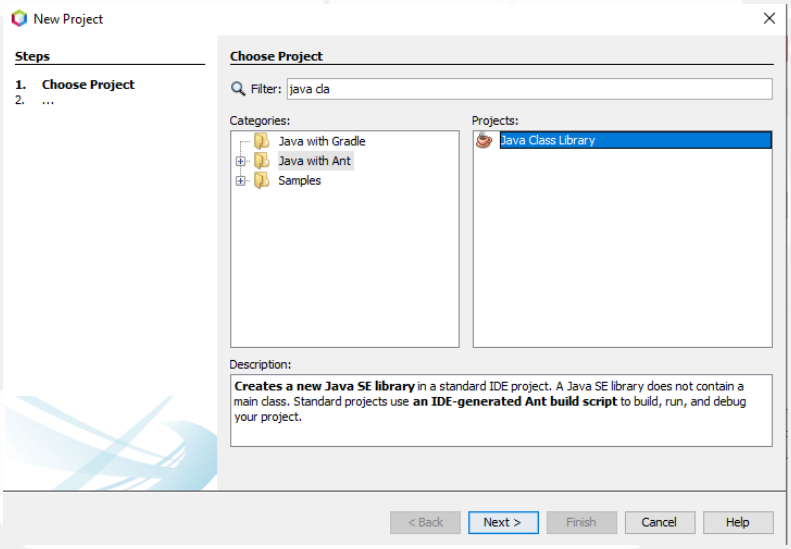
En este ejercicio se crearán casos de pruebas y conjunto de casos de prueba en Junit para un proyecto Java Class Library de Java with Ant. Para realizar esto se necesita el proyecto de prueba contenido en el archivo JUnitSample.zip disponible en el campus de la unidad

1. Creación del Proyecto

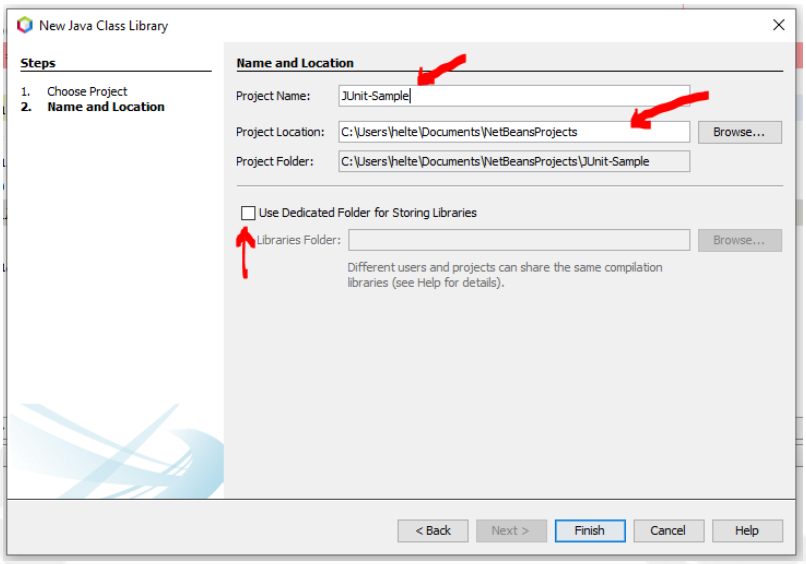
Primero crearemos un proyecto Java Class Library (librería de clases) llamado *Junit-Sample*. Después copiaremos 2 clases desde el proyecto de prueba descargado (JUnitSample) al proyecto creado.

Creación del Proyecto Java Class Library

1. En el menú principal, *File -> New Project*.
2. Selecciona *Java with Ant -> Java Class Library* y Click en *Next*.

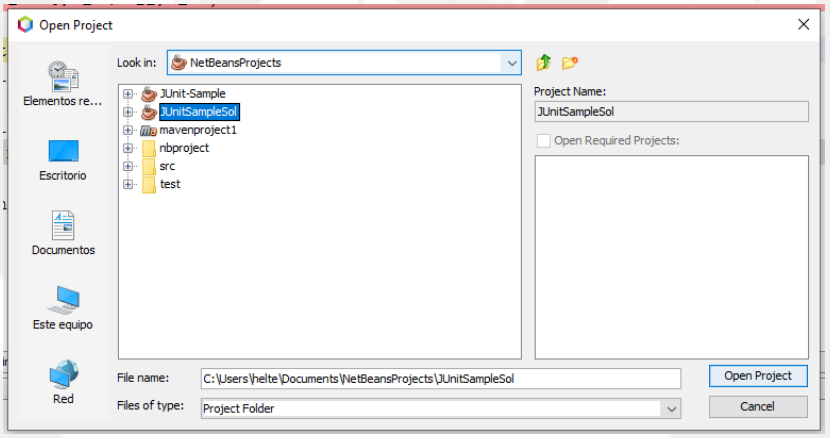


3. Escribe "*Junit-Sample*" como nombre del proyecto y especifica la ubicación del mismo.
4. Desmarca "*Use Dedicated Folder option*" en caso de que este marcado. Y pulsa en *Finish*.

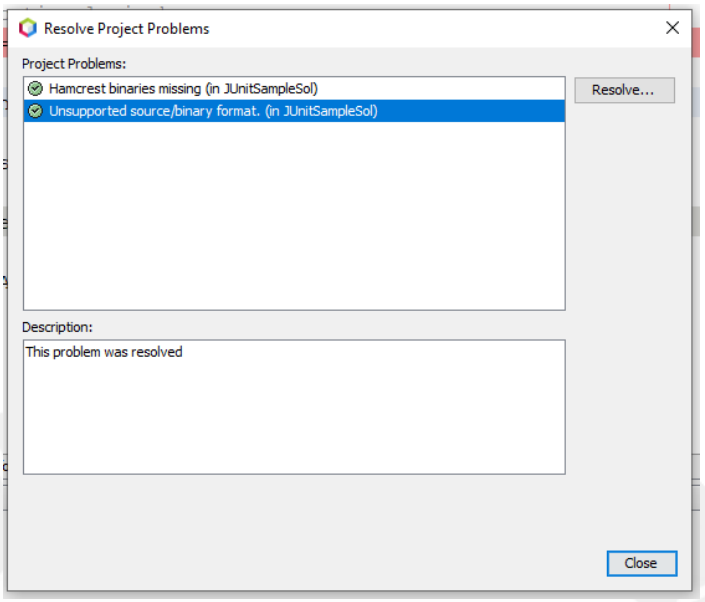


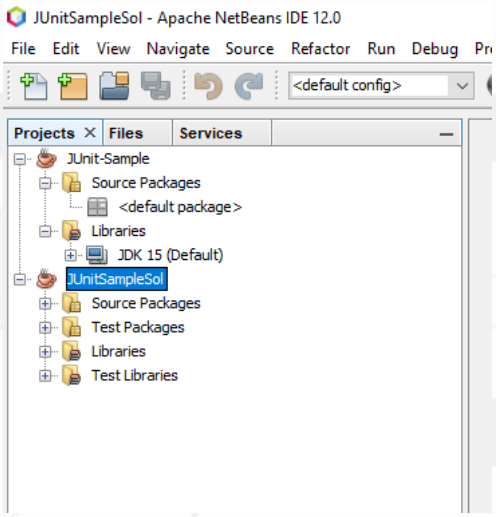
Descarga del Proyecto de prueba

1. Descárgate el proyecto de prueba y descomprímelo en tu carpeta de proyectos. A continuación, abre el proyecto. *File-> Open Project.*



En caso de que te aparezca un cuadro de dialogo que especifique que necesitas resolver problemas del proyecto, pulsa en *resolver*. Y soluciona todas las incidencias.

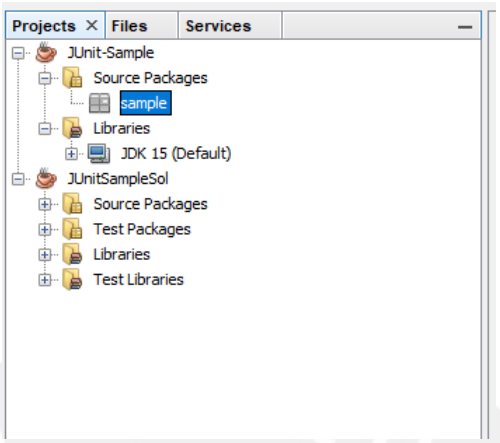




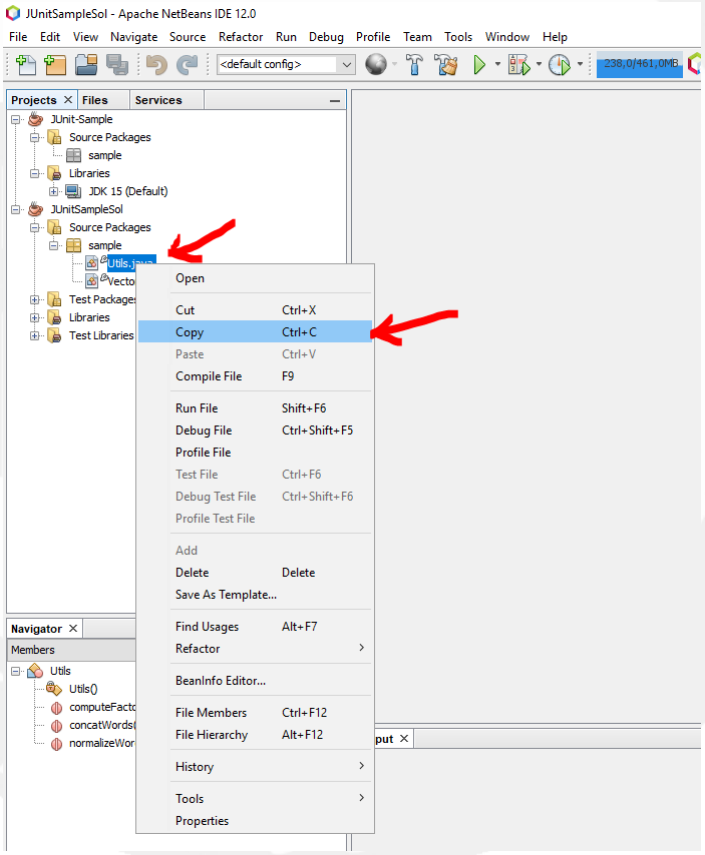
Creando las Clases Java

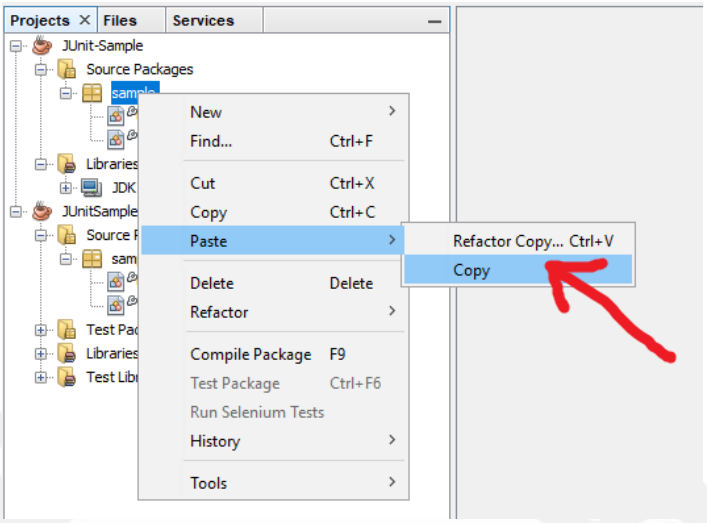
Ahora vamos a copiar los archivos *Utils.java* y *Vector.java* desde el proyecto de prueba al proyecto que hemos creado. *JUnitSampleSol -> Junit-Sample*.

1. En la venta de proyectos, con el click derecho en "*Source Packages*" de *Junit-Sample*, elegimos en el menú *New -> Java Package*.
2. Escribimos "*sample*" como nombre y le damos a *finish*.



3. Ahora expande el Source Packages de JUnitSampleSol y copia (botón derecho) Utils.java y Vectors.java desde sample al sample de Source Packages de Junit-Sample.





Ya puedes cerrar el Proyecto *JUnitSampleSol* porque no lo necesitaras. Aunque en el puedes encontrar todos los test que vamos a describir.

1. Si echas un vistazo a *Utils.java* verás que hay 3 metodos:
   a. *computeFactorial*, calcula el factorial de un número.
   b. *concatWords*, concatena cadenas de texto.
   c. *normalizedWord*, elimina caracteres diacríticos de una cadena (âäìîÛ…)

2. Y en *Verctors.java* verás 2 métodos;
   a. *equal*, comprueba que dos vectores sean iguales.
   b. *scalarMultiplication*, calcula el producto escalar de dos vectores.

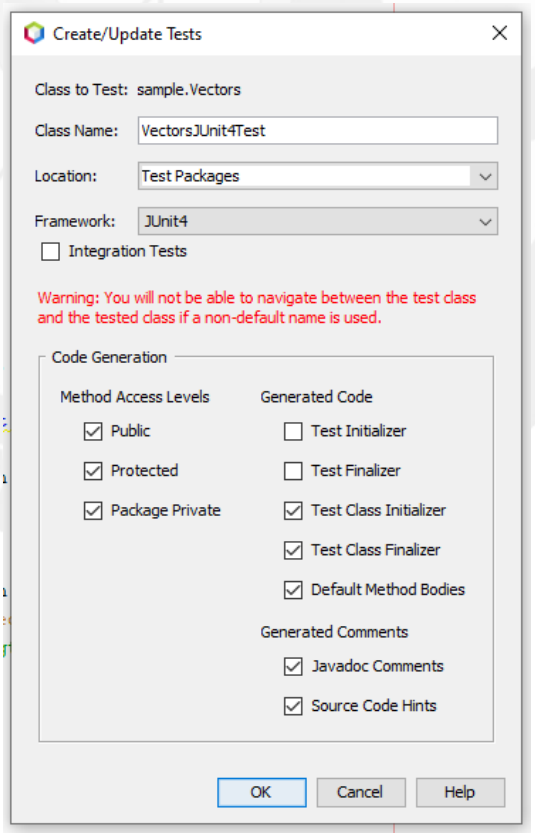
Lo siguiente que vamos a realizar es crear las clases de test y por cada clase y método escribir algunos Casos de Prueba.

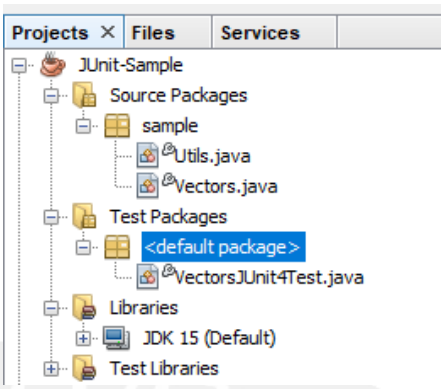
2. Creando Pruebas unitarias en Junit 4

Ahora vamos a crear clases de pruebas para las clases *Vector.java* y *Utils.java*. Vamos a utilizar el asistente de Netbeans para crear el esqueleto de las pruebas basado en las clases de nuestro proyecto.

Creando una Clase test para vector.java

1. Botón derecho sobre Vectors.java y elige Tools > Create Test.
2. Modifica el nombre de la clase de prueba a VectorsJUnit4Test en el cuadro de dialogo.
   Cuando cambias el nombre de la clase de prueba, verás una advertencia. El nombre por defecto está basado en el nombre de la clase que tú quieres probar, con la palabra Test como sufijo. Por ejemplo, la clase MyClass.java -> MyClassTest.java. Normalmente es mejor dejar el nombre por defecto, pero en este caso como estamos creando todos los test de JUnits en el mismo paquete los nombres de los test deben ser únicos.
3. Selecciona Junit4 en el framework.
4. Deselecciona Test Initializer y Test Finalizer. Pulsa Ok.





El IDE crea la clase de prueba VectorsJUnit4Test.java en el paquete por defecto dentro de Test Packages.

Un proyecto requiere un directorio de Test Packages para crear pruebas, el directorio por defecto se localiza en el root del proyecto, pero se puede especificar uno diferentes en la ventana de propiedades del proyecto.

Si echamos un vistazo a VectorsJUnit4Test.java en el editor, vemos que el IDE ha generado los métodos de test testEqual y testScalarMultiplication, uno por cada método que aparecía en la clase con el prefijo test (se puede cambiar el nombre), estos métodos tienen la anotación @Test.

El cuerpo de esos métodos de prueba ha sido generado a modo de guía y necesitaremos modificarlo según el caso de test. Si quisiéramos que no generara el cuerpo del método solo debemos desmarcar del cuadro de dialogo de creación de la clase de test, Default Method Bodies.

El IDE también genera los siguientes métodos Inicializadores y Finalizadores de la clase Test.

```
@BeforeClass
public static void setUpClass() throws Exception {
}

@AfterClass
public static void tearDownClass() throws Exception {
}
```

Estos métodos anotados con @BeforeClass y @AfterClass respectivamente son usados como métodos que deben ejecutarse antes y después de ejecutar la clase test. De momento no lo utilizaremos así que puedes comentarlos o borrarlos.

Escribiendo los métodos de prueba para Vectors.java

En este ejercicio vamos a modificar el test generado por el IDE con los métodos de la clase assert (aserciones o afirmaciones) que nos proporciona JUnit y cambiaremos el nombre de los métodos (Junit4 es flexible con el nombre de los métodos, pero es necesario que lleven la anotación @Test).

1. Abrimos en el editor VectorsJUnit4Test.java.

2. Modificamos el nombre del método testScalarMultiplication, el valor del println y borramos las variables generadas. Quedando de la siguiente manera.

```
@Test
public void ScalarMultiplicationCheck() {
   System.out.println("* VectorsJUnit4Test: ScalarMultiplicationCheck()");
   assertEquals(expResult, result);
}
```

3. Ahora añadimos los casos de prueba con algunos asserts. Los más comunes son los siguientes:

- assertTrue/assertFalse (condición a testear). Comprueba que la condición es cierta o falsa.
- assertEquals/assertNotEquals (valor esperado, valor obtenido). Es importante el orden de los valores esperado y obtenido.
- assertNull/assertNotNull (object). Comprueba que el objeto obtenido es nulo o no.
- assertSame/assertNotSame(object1, object2). Comprueba si dos objetos son iguales o no.
- fail(). Fuerza que el test termine con fallo. Se puede indicar un mensaje.

Al final de este documento tienes enlaces donde podrás ver que métodos tiene la clase asserts.

El código quedará asi:

```
@Test
public void ScalarMultiplicationCheck() {
   System.out.println("* VectorsJUnit4Test: ScalarMultiplicationCheck()");
   assertEquals(  0, Vectors.scalarMultiplication(new int[] { 0, 0}, new int[] { 0, 0}));
   assertEquals( 39, Vectors.scalarMultiplication(new int[] { 3, 4}, new int[] { 5, 6}));
   assertEquals(-39, Vectors.scalarMultiplication(new int[] {-3, 4}, new int[] { 5,-6}));
   assertEquals(  0, Vectors.scalarMultiplication(new int[] { 5, 9}, new int[] {-9, 5}));
   assertEquals(100, Vectors.scalarMultiplication(new int[] { 6, 8}, new int[] { 6, 8}));
}
```

En este método de prueba ScalarMultiplicationCheck(), hemos utilizado el método de la clase assert “assertEquals“ donde el primer parámetro es el valor esperado

"expResult" y el segundo parámetro es el parámetro obtenido (técnica de caja negra). Para pasar el Test, el método de prueba debe tener éxito en todos los valores esperados. Debemos añadir suficientes números de assert para cubrir los posibles casos de prueba.

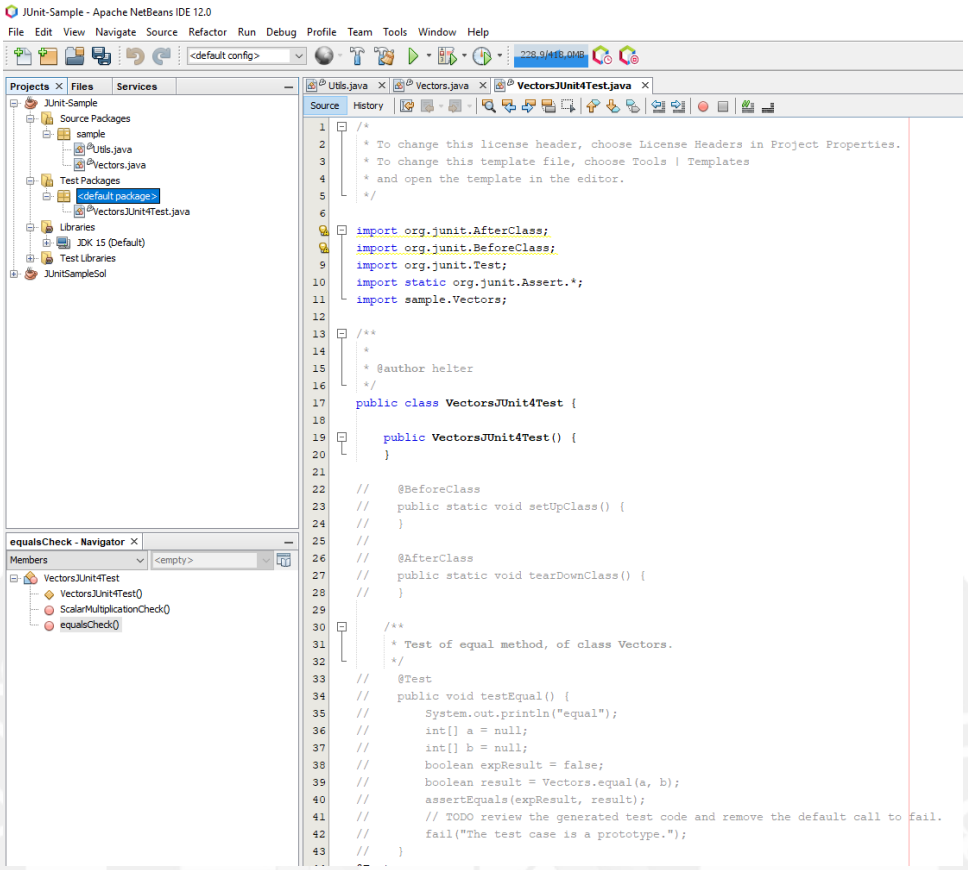
4. Ahora realizaremos lo mismo con el método testEqual, deberá quedar así (no olvides cambiar el nombre):

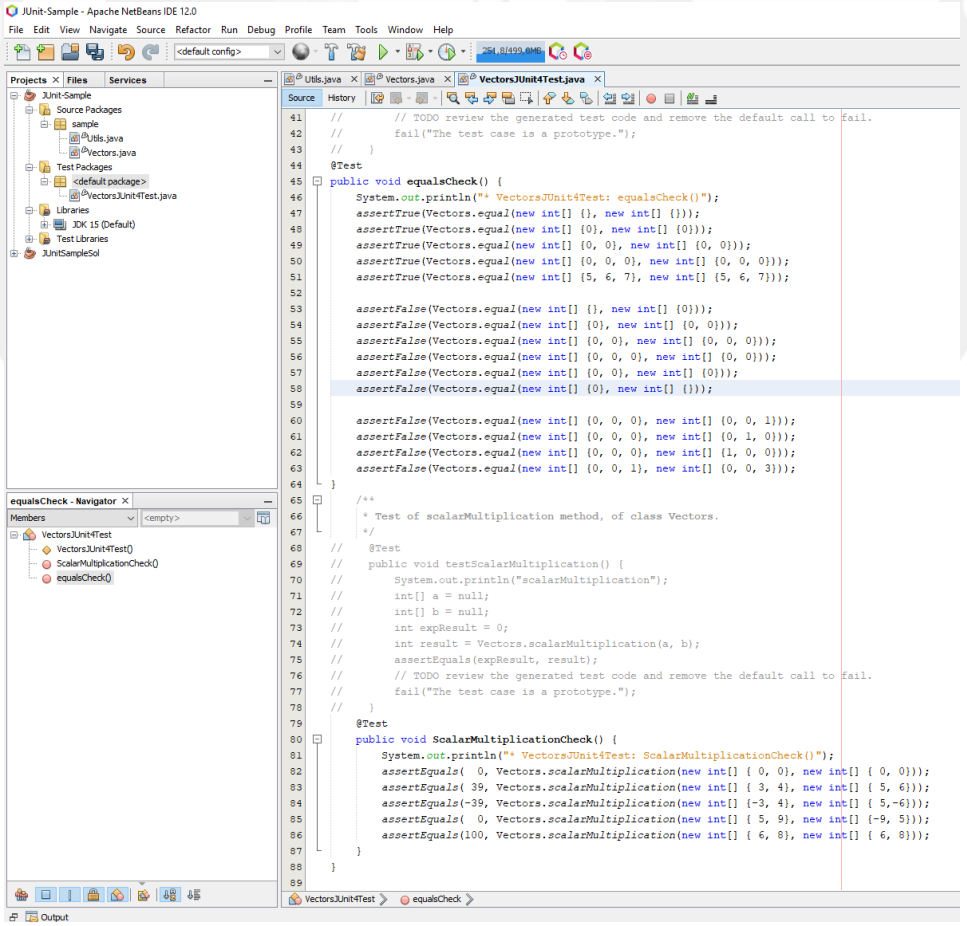
```
@Test
public void equalsCheck() {
   System.out.println("* VectorsJUnit4Test: equalsCheck()");
   assertTrue(Vectors.equal(new int[] {}, new int[] {}));
   assertTrue(Vectors.equal(new int[] {0}, new int[] {0}));
   assertTrue(Vectors.equal(new int[] {0, 0}, new int[] {0, 0}));
   assertTrue(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0, 0}));
   assertTrue(Vectors.equal(new int[] {5, 6, 7}, new int[] {5, 6, 7}));

   assertFalse(Vectors.equal(new int[] {}, new int[] {0}));
   assertFalse(Vectors.equal(new int[] {0}, new int[] {0, 0}));
   assertFalse(Vectors.equal(new int[] {0, 0}, new int[] {0, 0, 0}));
   assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0}));
   assertFalse(Vectors.equal(new int[] {0, 0}, new int[] {0}));
   assertFalse(Vectors.equal(new int[] {0}, new int[] {}));

   assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0, 1}));
   assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 1, 0}));
   assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {1, 0, 0}));
   assertFalse(Vectors.equal(new int[] {0, 0, 1}, new int[] {0, 0, 3}));
}
```

Para que el test de este método ahora tenga éxito y se considere pasado, los assertTrue deben ser verdaderos y los assertFalse deben ser falsos.


JUnit-Sample - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
JUnit-Sample
Source Packages
sample
Utils.java
Vectors.java
Test Packages
<default package>
VectorsJUnit4Test.java
Libraries
JDK 15 (Default)
Test Libraries
JUnitSampleSol
equalsCheck - Navigator
Members
<empty>
VectorsJUnit4Test
VectorsJUnit4Test()
ScalarMultiplicationCheck()
equalsCheck()
Utils.java
Vectors.java
VectorsJUnit4Test.java
Source History
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import sample.Vectors;
/**
*
* @author helter
*/
public class VectorsJUnit4Test {
public VectorsJUnit4Test() {
}
// @BeforeClass
// public static void setUpClass() {
// }
//
// @AfterClass
// public static void tearDownClass() {
// }
/**
* Test of equal method, of class Vectors.
*/
// @Test
// public void testEqual() {
// System.out.println("equal");
// int[] a = null;
// int[] b = null;
// boolean expResult = false;
// boolean result = Vectors.equal(a, b);
// assertEquals(expResult, result);
// // TODO review the generated test code and remove the default call to fail.
// fail("The test case is a prototype.");
// }


JUnit-Sample - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
JUnit-Sample
Source Packages
sample
Utils.java
Vectors.java
Test Packages
<default package>
VectorsJUnit4Test.java
Libraries
JDK 15 (Default)
Test Libraries
JUnitSampleSol
equalsCheck - Navigator
Members
<empty>
VectorsJUnit4Test
VectorsJUnit4Test()
ScalarMultiplicationCheck()
equalsCheck()
Utils.java
Vectors.java
VectorsJUnit4Test.java
Source History
// // TODO review the generated test code and remove the default call to fail.
// fail("The test case is a prototype.");
// }
@Test
public void equalsCheck() {
System.out.println("* VectorsJUnit4Test: equalsCheck()");
assertTrue(Vectors.equal(new int[] {}, new int[] {}));
assertTrue(Vectors.equal(new int[] {0}, new int[] {0}));
assertTrue(Vectors.equal(new int[] {0, 0}, new int[] {0, 0}));
assertTrue(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0, 0}));
assertTrue(Vectors.equal(new int[] {5, 6, 7}, new int[] {5, 6, 7}));
assertFalse(Vectors.equal(new int[] {}, new int[] {0}));
assertFalse(Vectors.equal(new int[] {0}, new int[] {0, 0}));
assertFalse(Vectors.equal(new int[] {0, 0}, new int[] {0, 0, 0}));
assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0}));
assertFalse(Vectors.equal(new int[] {0, 0}, new int[] {0}));
assertFalse(Vectors.equal(new int[] {0}, new int[] {}));
assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0, 1}));
assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 1, 0}));
assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {1, 0, 0}));
assertFalse(Vectors.equal(new int[] {0, 0, 1}, new int[] {0, 0, 3}));
}
/**
* Test of scalarMultiplication method, of class Vectors.
*/
// @Test
// public void testScalarMultiplication() {
// System.out.println("scalarMultiplication");
// int[] a = null;
// int[] b = null;
// int expResult = 0;
// int result = Vectors.scalarMultiplication(a, b);
// assertEquals(expResult, result);
// // TODO review the generated test code and remove the default call to fail.
// fail("The test case is a prototype.");
// }
@Test
public void ScalarMultiplicationCheck() {
System.out.println("* VectorsJUnit4Test: ScalarMultiplicationCheck()");
assertEquals(0, Vectors.scalarMultiplication(new int[] { 0, 0}, new int[] { 0, 0}));
assertEquals(39, Vectors.scalarMultiplication(new int[] { 3, 4}, new int[] { 5, 6}));
assertEquals(-39, Vectors.scalarMultiplication(new int[] {-3, 4}, new int[] { 5,-6}));
assertEquals(0, Vectors.scalarMultiplication(new int[] { 5, 9}, new int[] {-9, 5}));
assertEquals(100, Vectors.scalarMultiplication(new int[] { 6, 8}, new int[] { 6, 8}));
}
}
VectorsJUnit4Test
equalsCheck
Output

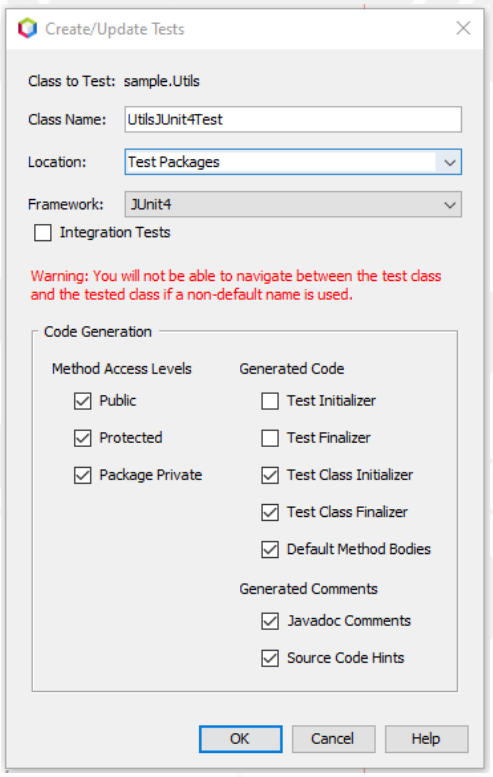
Creando una clase test para utils.java

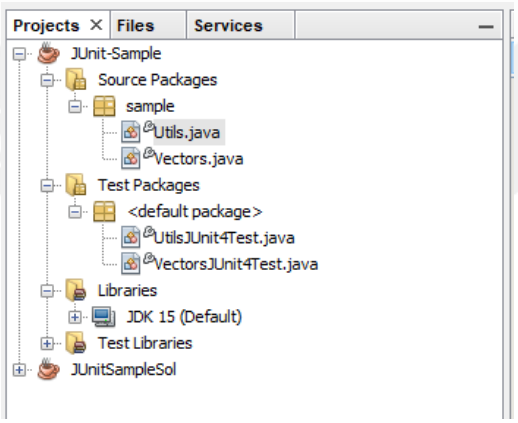
1. Botón derecho sobre *Utilss.java* y elige *Tools* > *Create Test*.

2. Modifica el nombre de la clase de prueba a *UtilsJUnit4Test* en el cuadro de dialogo.

3. Selecciona Junit4 en el framework.

4. Deselecciona *Test Initializer* y *Test Finalizer*. Pulsa *Ok*.





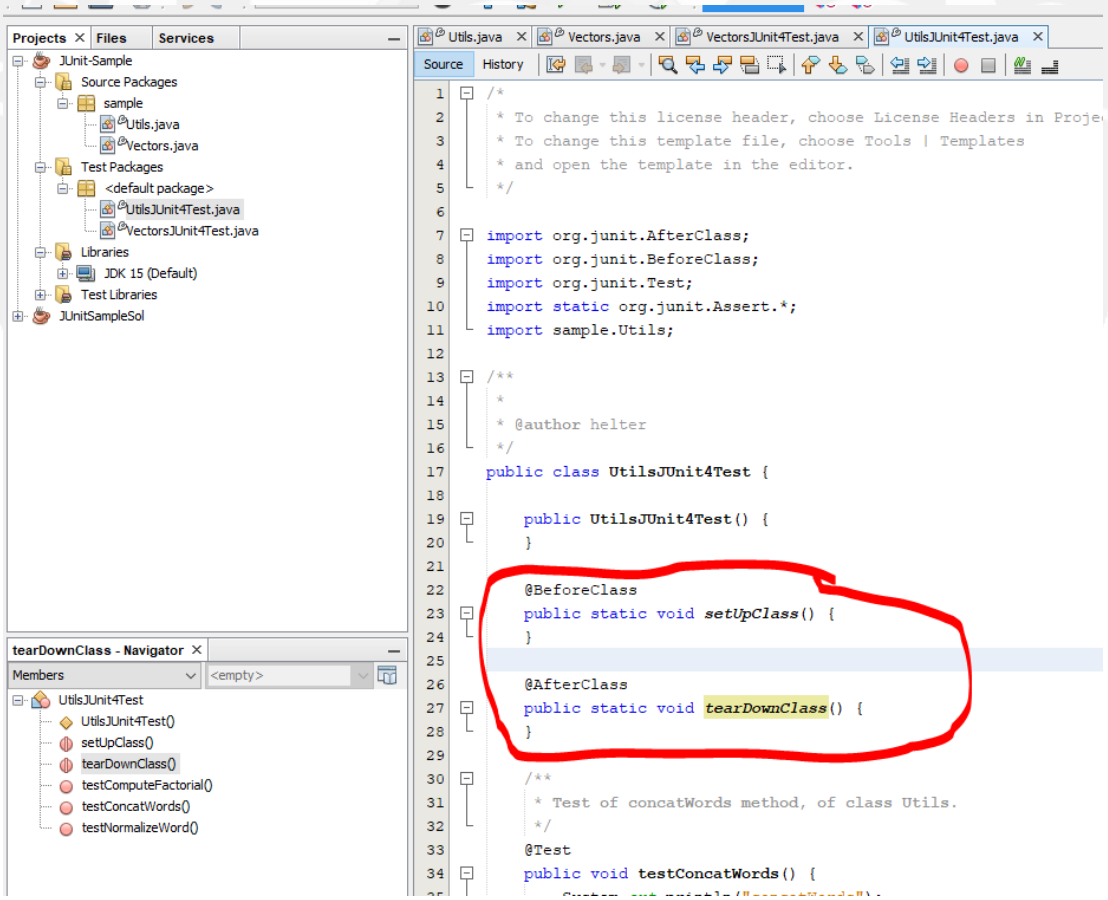
El IDE crea la clase de prueba *UtilsJUnit4Test.java* en el paquete por defecto dentro de *Test Packages*. Vemos que el IDE ha generado los métodos de test *testComputeFactorial*, *testConcatWords* y *testNormalizeWord*, uno por cada método de *Utils.java*. El IDE también genera los métodos Inicializadores y Finalizadores de la clase Test.

Escribiendo los métodos de prueba para Utils.java

En este ejercicio añadiremos Casos de Prueba que ilustrarán algunos de los elementos comunes de los Test con Junit. También añadiremos un println a los métodos porque algunos métodos no muestran ninguna salida por la ventana de Resultados de Test y necesitaremos indicar que se han ejecutado o que han pasado el test, además esto nos posibilitara ver el orden en el que se han ejecutado las pruebas.

Pruebas con Inicializadores y Finalizadores

Cuando creamos la clase Test para Utils.java el IDE generó los métodos inicializador y finalizador (con las anotaciones; @BeforeClass y @AfterClass), en nuestro caso es probable que con los nombres setUpClass() y tearDownClass() estos nombres de métodos se pueden cambiar por el que queramos los importante son las anotaciones.



Como dijimos en el ejercicio del test de la clase vector estos métodos inicializadores y finalizadores en realidad no son necesarios, pero en este ejercicio se busca ilustrar su funcionamiento.

Así pues, en Junit 4 podemos utilizar anotaciones para marcar los siguientes tipos de métodos inicializadores y finalizadores:

- **Inicializador de la Clase Test.** (@BeforeClass). Esta anotación marca un método que se ejecutara solo una vez antes de cualquier método de la clase. Por ejemplo, en vez de crear una conexión a una base de datos al iniciar cada uno de los métodos de prueba (con un inicializador de prueba @Before) y crear una nueva conexión. Podríamos abrir una conexión a la base de datos aquí y cuando se terminen de ejecutar todos los test podríamos cerrarla en el finalizador de clase (@AfterClass).
- **Finalizador de la Clase Test.** (@AfterClass). El finalizador se ejecuta solo una vez después de que todos los métodos de prueban hayan finalizado.
- **Inicializador de la Prueba.** (@Before). Este inicializador de la prueba se ejecuta antes de que cada método de prueba en la clase test. No es necesario para ejecutar la prueba, pero si necesitamos, por ejemplo, inicializar algunas variables antes de ejecutar la prueba podremos hacerlo en este método.
- **Finalizador de la Prueba.** (@After). Este finalizador de la prueba se ejecuta después de que cada método de prueba en la clase test. No es necesario para ejecutar la prueba, pero si necesitamos por ejemplo limpiar o liberar cualquier dato que hemos necesitado para ejecutar el caso de prueba podremos hacerlo aquí.

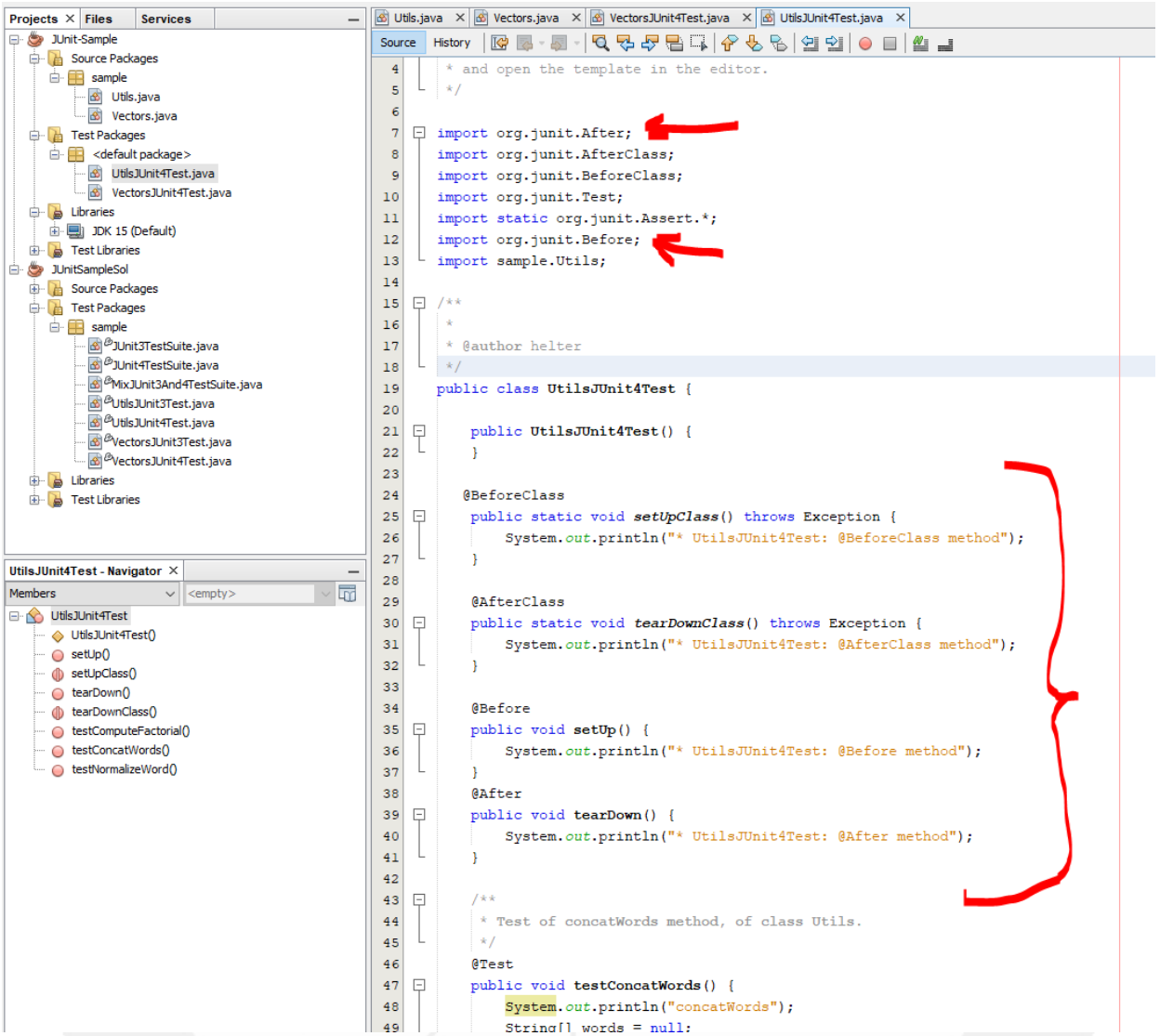
Vamos a realizar los siguientes cambios en el código en UtilsJUnit4Test.java:

```
@BeforeClass
public static void setUpClass() throws Exception {
   System.out.println("* UtilsJUnit4Test: @BeforeClass method");
}

@AfterClass
public static void tearDownClass() throws Exception {
   System.out.println("* UtilsJUnit4Test: @AfterClass method");
}

@Before
public void setUp() {
   System.out.println("* UtilsJUnit4Test: @Before method");
}

@After
public void tearDown() {
   System.out.println("* UtilsJUnit4Test: @After method");
}
```



Necesitaremos importar After y Before.

Cuando ejecutemos esta Clase Test se mostrarán los println por la ventana de Resultados de Test de Junit. Si no ponemos esos println nada indicará que los inicializadores y finalizadores se han ejecutado.

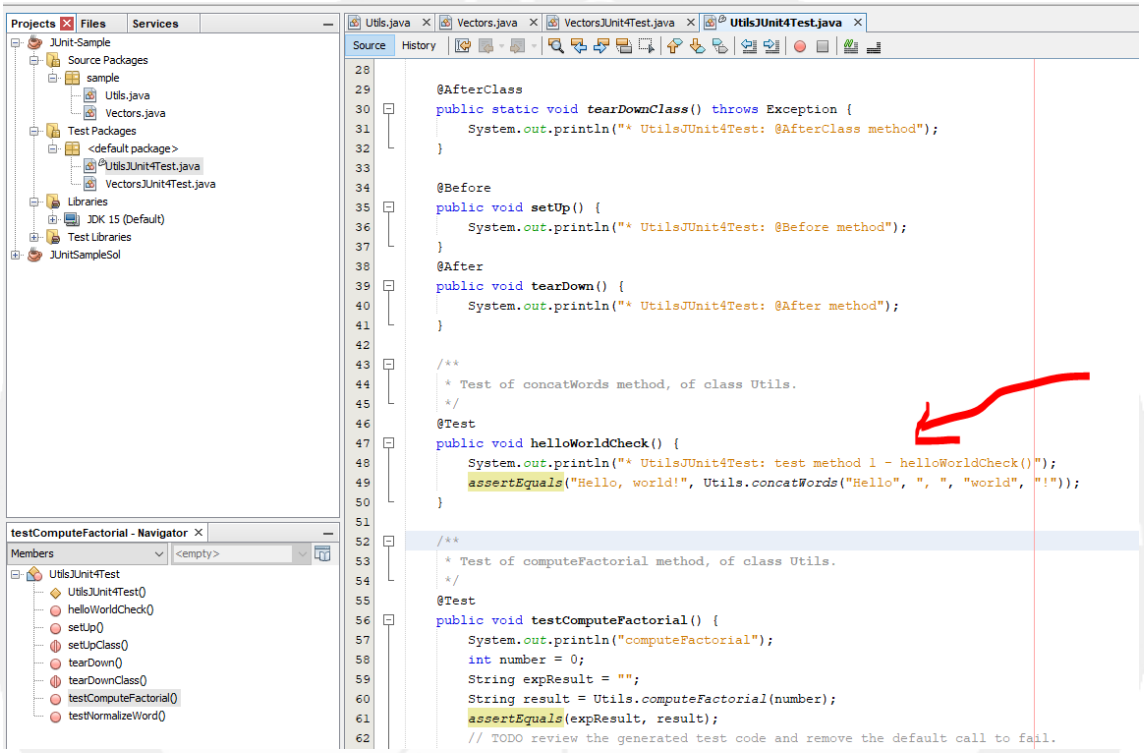
Prueba usando afirmación (assertion) simple

Realizaremos un caso de prueba simple para el método de prueba de concatWords. En vez de usar el método de prueba generado testConcatWords, añadiremos un nuevo método de prueba llamado helloWorldCheck que usara una assertion (afirmación) simple para probar si el método concatena las cadenas de texto correctamente. AssertEquals en el caso de prueba tiene la sintaxis assertEquals(EXPECTED_RESULT, ACTUAL_RESULT) para probar que el resultado esperado es igual al resultado actual.. En este caso, si la entrada al metodo concatWords es " Hello ", " , ", " world " y " ! ", da como resultado "*Hello, world!*".

1. Borramos el método testConcatWords.

2. Añadimos el siguiente:

```
@Test
public void helloWorldCheck() {
        System.out.println("* UtilsJUnit4Test: test method 1 - helloWorldCheck()");
        assertEquals("Hello, world!", Utils.concatWords("Hello", ", ", "world", "!"));
}
```



Pruebas usando Timeout

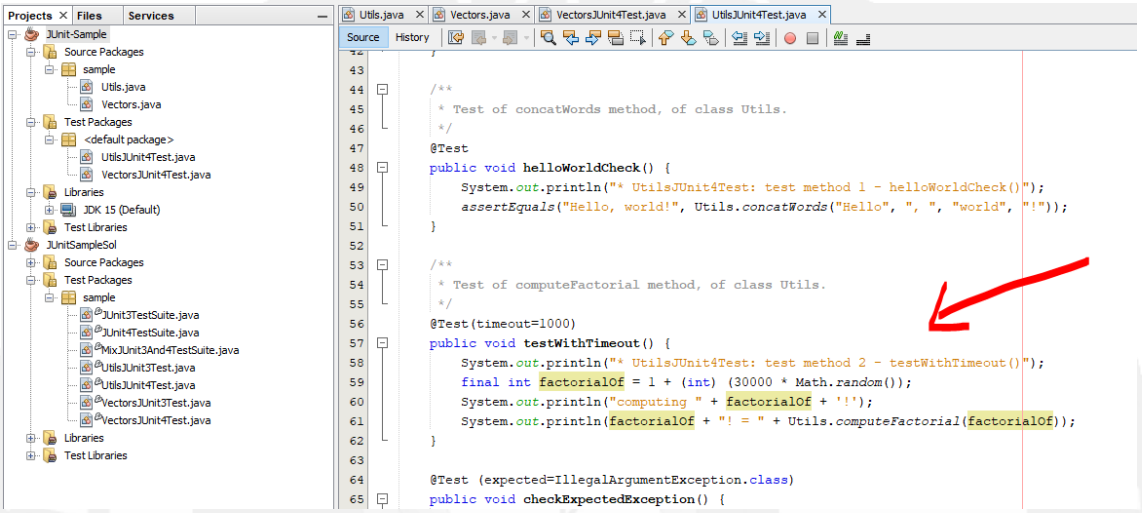
Esta prueba quiere demostrar como comprobar si a un método le está llevando mucho tiempo realizar los cálculos. Si el método esta tardando mucho, la prueba se interrumpe y da fallo. Para ello especificaremos un tiempo límite en la prueba.
El método de la prueba invoca al método computeFactorial de Utils.java. Podemos asumir que el método computeFactorial es correcto, pero en este caso queremos comprobar si la computación del método tarda 1000 milisegundos, haciendo que si tarda más se interrumpa la prueba. Cuando se interrumpe la prueba se lanza una Excepción de TimeoutException.

1. Borra el método de test que ha generado el IDE, testcomputeFactorial.

2. Añade el método testWithTimeout que calculará el factorial de un número aleatorio generado.

```
@Test (timeout=1000)
public void testWithTimeout() {
        System.out.println("* UtilsJUnit4Test: test method 2 - testWithTimeout()");
        final int factorialOf = 1 + (int) (30000 * Math.random());
        System.out.println("computing " + factorialOf + '!');
        System.out.println(factorialOf + "! = " + Utils.computeFactorial(factorialOf));
}
```

Observa que hemos añadido el timeout de 1000.



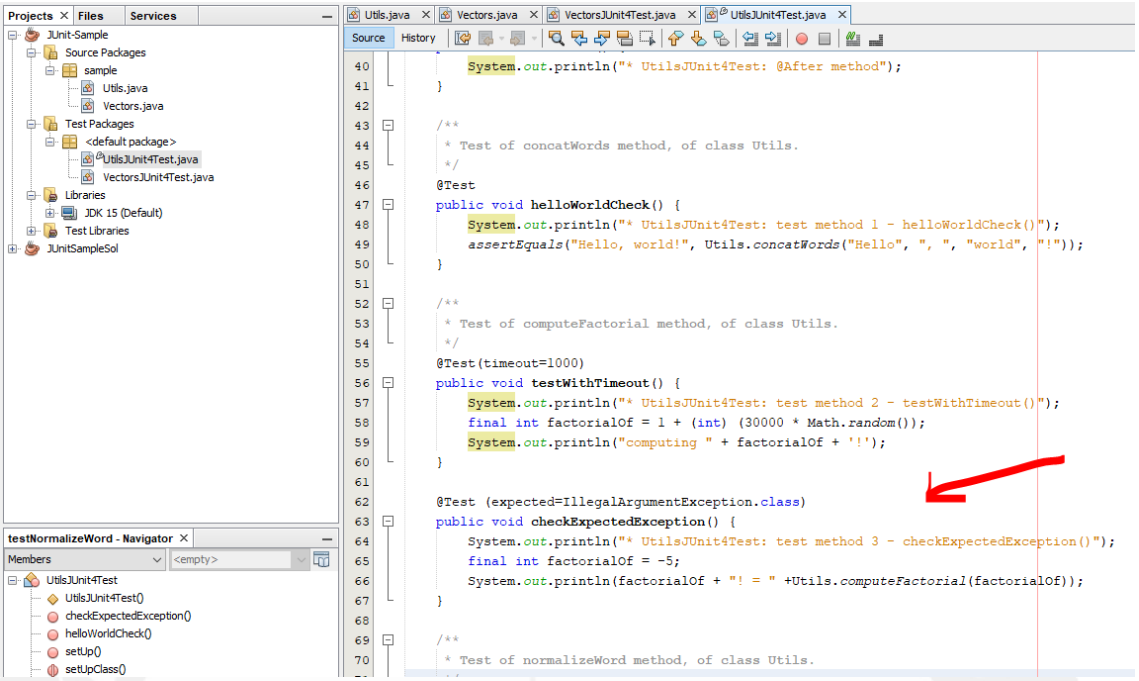
Pruebas de esperando una excepción

Esta prueba muestra cómo realizar un caso de prueba que espere que se lance una excepción. Es decir, el método falla si no se lanza la Excepción esperada. En este caso vamos a probar el método computeFactorial que lanzará una excepción IllegalArgumentException cuando pongamos un parámetro de entrada negativo (-5).

1. Añade el siguiente el siguiente método:

```
@Test (expected=IllegalArgumentException.class)
public void checkExpectedException() {
System.out.println("* UtilsJUnit4Test: test method 3 - checkExpectedException()");
final int factorialOf = -5;
System.out.println(factorialOf + "! = " +Utils.computeFactorial(factorialOf));
}
```

Observa que hemos añadido expected=IllegalArgumentException.class



Deshabilitando una Prueba

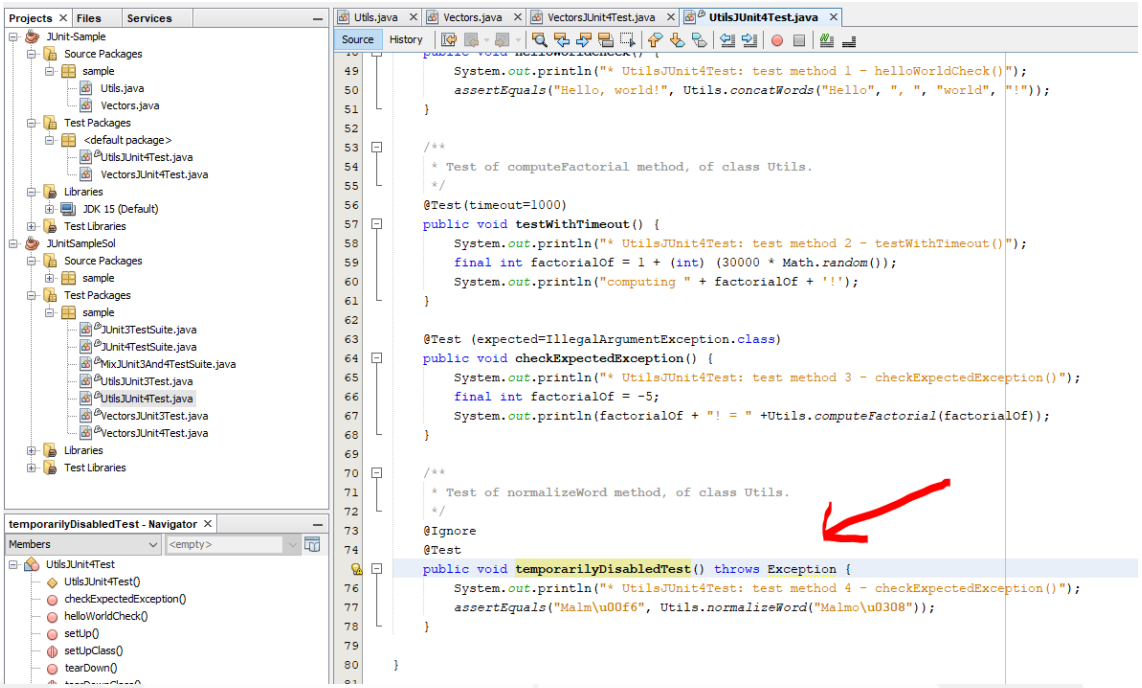
Esta prueba muestra cómo podemos deshabilitar una prueba temporalmente. En Junit4 simplemente se añade la anotación @Ignore al texto a deshabilitar.

1. Borra el método de prueba generado por el IDE testNormalizeWord.

2. Añade el siguiente método:

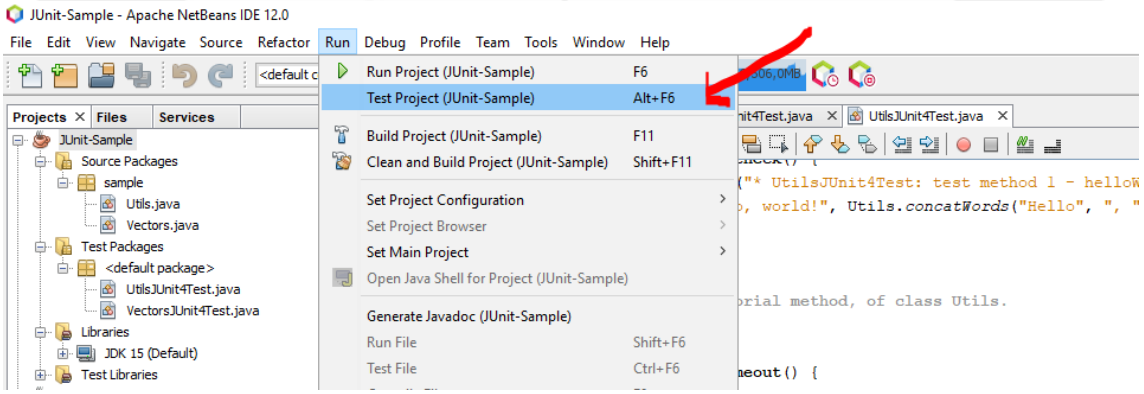
```
@Test
public void temporarilyDisabledTest() throws Exception {
    System.out.println("* UtilsJUnit4Test: test method 4 - checkExpectedException()");
    assertEquals("Malm\u00f6", Utils.normalizeWord("Malmo\u0308"));
}
```

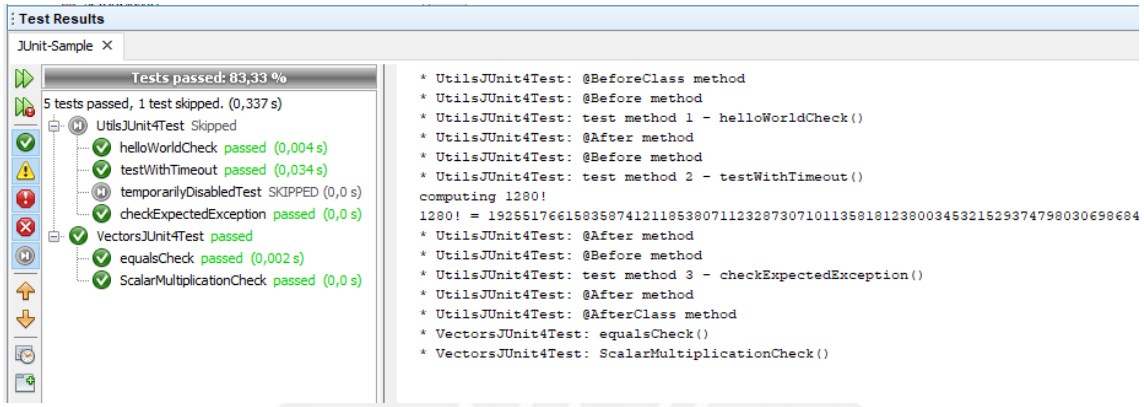
Añade la anotación @Ignore encima de la anotación @Test para deshabilitar esa Prueba. (Tendrás que importar org.junit.Ignore).



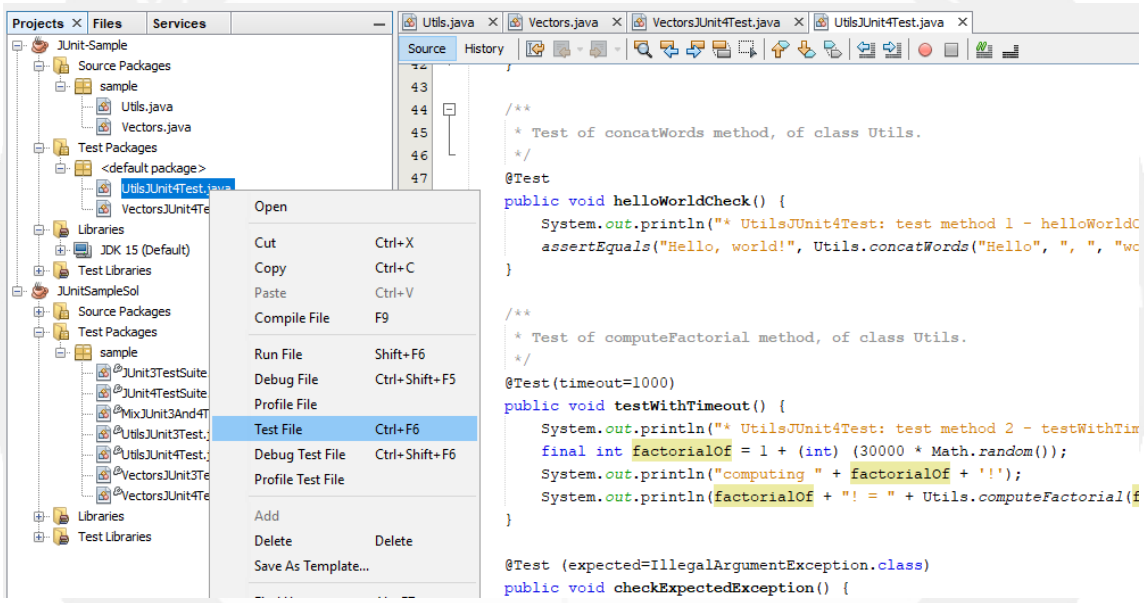
3. Ejecución de las pruebas

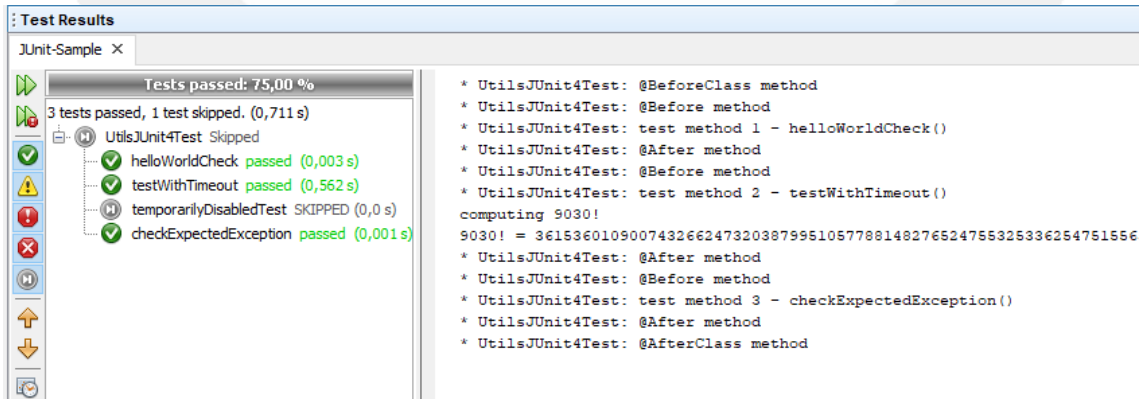
Podemos ejecutar las pruebas Junit para toda la aplicación o para cada uno de los archivos y podemos ver los resultados en el IDE. La forma más fácil de ejecutar todas las pruebas del proyecto eligiendo en el menú principal Run>Test Project (Junit-Sample). Con esta opción se ejecutarán todos los métodos de todas las clases Test de TextPackages.





Para ejecutar una clase test individualmente, haz click derecho sobre la clase en Test Packages y elige Test File.





En la ventana de Resultados del Test podemos ver que han pasado todas las pruebas. En el panel izquierdo podemos ver el resultado de las pruebas individualmente, en este panel podemos relanzar una prueba o ir al código fuente, y en el panel de la derecha se

muestra todas las salidas que hemos añadido a los test. Si miramos con detenimiento este panel podemos ver el orden en el que se han ejecutados cada uno de los test. Como puedes ver los inicializadores y finalizadores de clase (@BeforeClass y @AfterClass) se ejecutan al comienzo y al final de toda la clase Test, y cada inicializador y finalizador de método (@Before y @After) cada vez que se ejecuta un método.

4. Creando Suites de Prueba

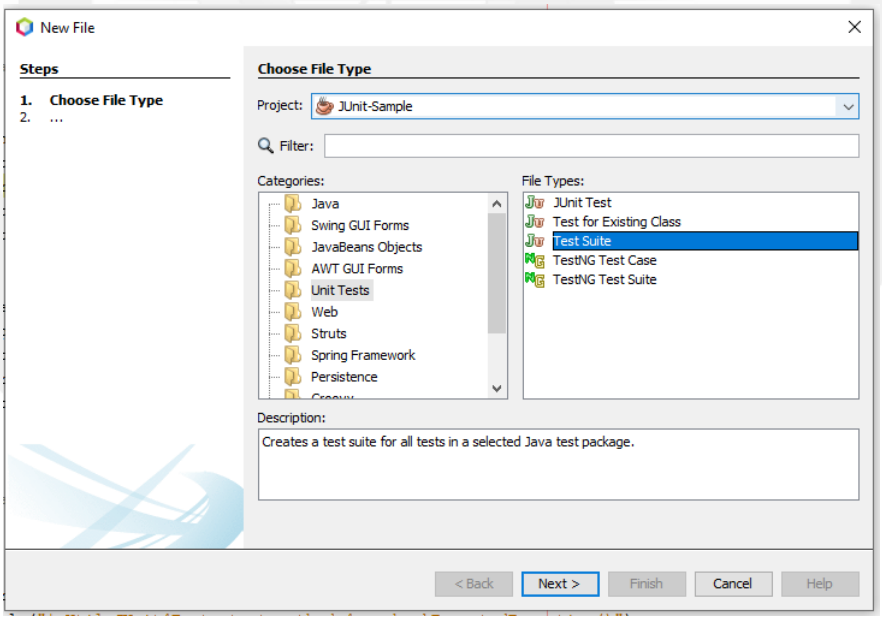
Cuando creamos pruebas para un proyecto podemos terminar literalmente con un montón de Clases de Test. En muchas ocasiones tu querrás lanzar un subconjunto de estas pruebas o lanzar las pruebas en un orden especifico. Para esto se puede crear una o mas Suite de Pruebas.

Un Suite de pruebas es básicamente una clase con un método que invoca casos de prueba, Clases de test o métodos de prueba específicos.

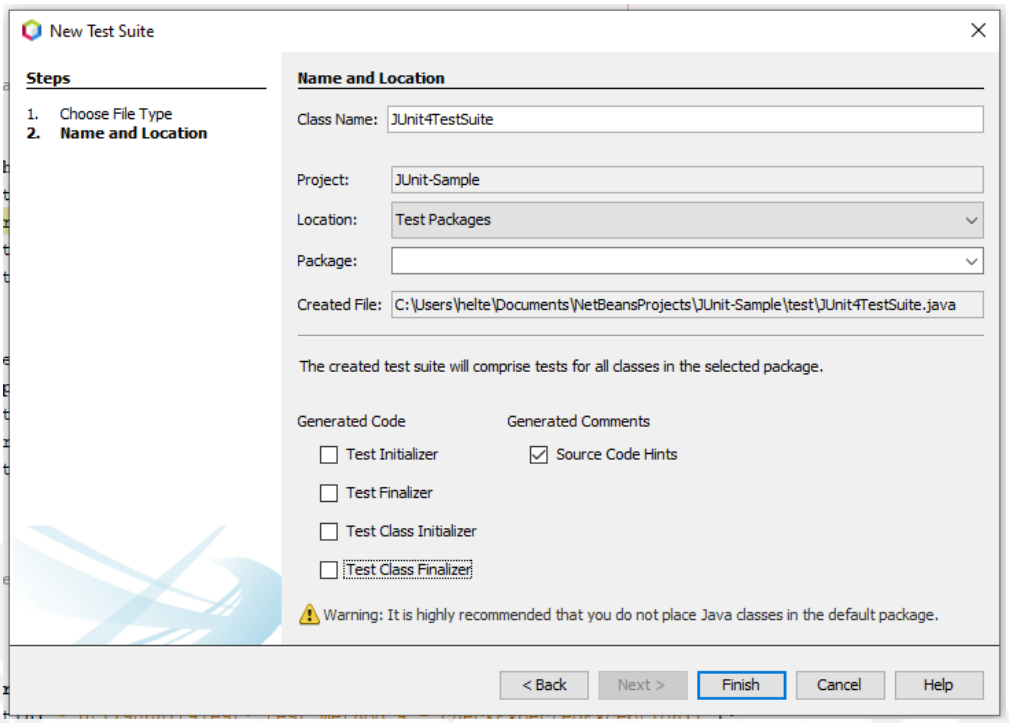
Podemos crear Suites de prueba de manera manual o automáticamente con el IDE. Cuando lo hacemos de forma automática el IDE genera código que invocara a todas las clases de test del mismo paquete que posteriormente podremos modificarlo para especificar que pruebas queremos ejecutar y cuales no en esa suite.

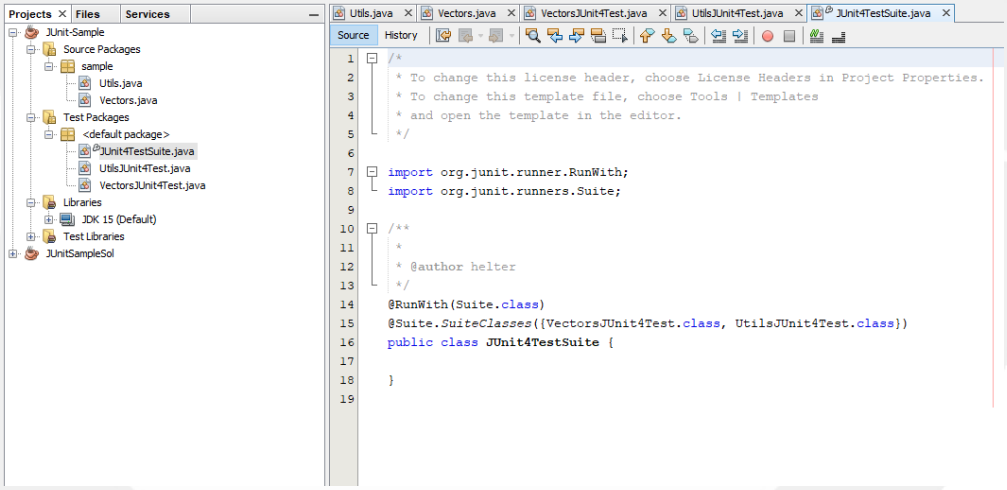
Creando y ejecutando Suites de pruebas

1. Botón derecho sobre el proyecto y elige New > Other, se abrirá el asistente.
2. Seleccionamos Test Suites en la categoría de Test unitarios, y pulsamos Next.



3. Elegimos un nombre para nuestro Test Suite, en este caso NewTestSuite
4. Elegimos los paquetes de Test y desmarcamos Test Class initializer y finalizer.
5. Pulsamos sobre Finish.





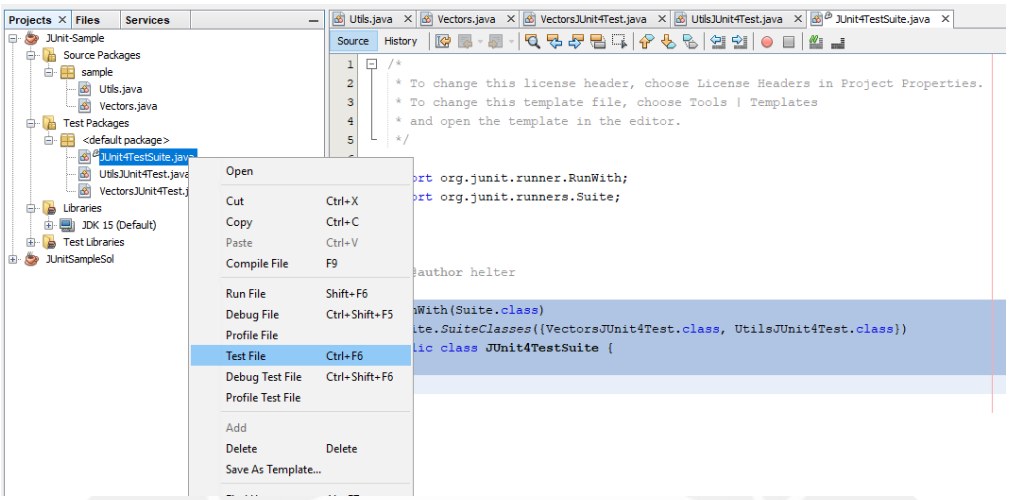
Como vemos se ha generado el código:

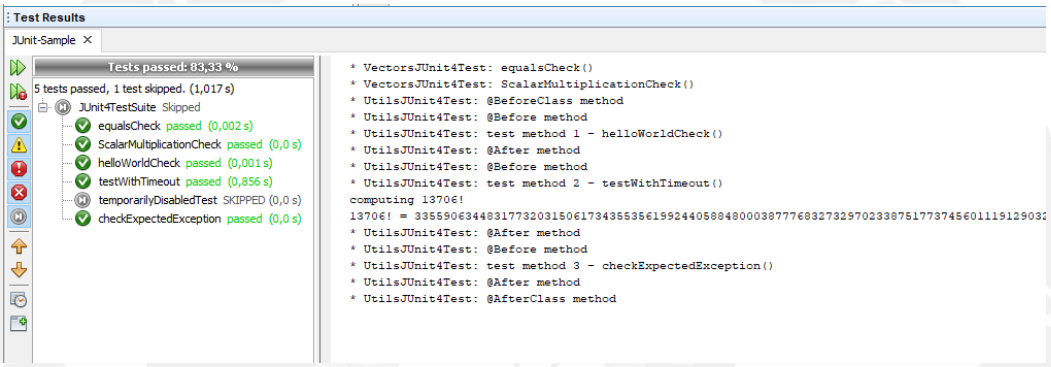
```
@RunWith(Suite.class)
@Suite.SuiteClasses({VectorsJUnit4Test.class, UtilsJUnit4Test.class})
public class JUnit4TestSuite {
}
```

Cuando se ejecute esta suite se ejecutarán las clases de test en el orden que se ha especificado aquí, obsérvalo en la anotación @Suite.SuiteClasses({VectorsJUnit4Test.class, UtilsJUnit4Test.class}).

Para Ejecutar el Suite de Prueba:

1. Selecciónalo en árbol del proyecto y con el botón derecho elige Test File.





5. Enlaces de Interés

Métodos de la Clase Assert de Junit4
http://junit.sourceforge.net/javadoc/index.html?org/junit/Assert.html
https://github.com/junit-team/junit4/wiki/Assertions

Junit sobre proyectos Maven
https://danielme.com/2016/09/07/tutorial-testing-con-junit-4/