- Crear el proyecto
    - Creación del proyecto de biblioteca de clases de Java
- Crear las clases de Java
- Escritura de pruebas unitarias JUnit 3
    - Creación de una clase de prueba paraVectors.java
    - Escribir métodos de prueba paraVectors.java
    - Creación de una clase de prueba paraUtils.java
    - Escribir métodos de prueba paraUtils.java
    - Ejecución de las pruebas
- Escritura de pruebas JUnit 4
    - Creación de una clase de prueba paraVectors.java
    - Escribir métodos de prueba paraVectors.java
    - Creación de una clase de prueba paraUtils.java
    - Escribir métodos de prueba paraUtils.java
    - Ejecución de las pruebas
- Creación de conjuntos de pruebas
    - Creación de conjuntos de pruebas JUnit 3
    - Creación de conjuntos de pruebas JUnit 4
    - Ejecución de conjuntos de pruebas
- Conclusión

Este tutorial presenta los conceptos básicos para escribir y ejecutar pruebas unitarias JUnit en NetBeans IDE. Probar una aplicación es una parte integral del ciclo de desarrollo, y escribir y mantener pruebas unitarias puede ayudar a garantizar que los métodos individuales en su código fuente funcionen correctamente. El soporte integrado del IDE para el marco de pruebas de unidades JUnit le permite crear rápida y fácilmente pruebas y conjuntos de pruebas JUnit.

En este tutorial, creará pruebas unitarias JUnit 3 y JUnit 4 simples y suites de prueba para un proyecto de biblioteca de clases Java. La primera parte del tutorial muestra cómo crear pruebas en JUnit 3. La segunda parte muestra cómo crear las mismas pruebas en JUnit 4 usando anotaciones JUnit. No es necesario completar ambas partes del tutorial porque las pruebas son las mismas, pero ver cómo se escriben las pruebas en ambas versiones le permite ver algunos de los cambios introducidos en JUnit 4.

Para obtener más información sobre el uso de JUnit, consulte www.junit.org .

Crear el proyecto

Para completar este tutorial, primero debe crear un proyecto de biblioteca de clases de Java llamado JUnit-Sample. Después de crear el proyecto, copie dos clases del proyecto de muestra JUnitSampleSol a su proyecto JUnit-Sample.

Creación del proyecto de biblioteca de clases de Java

1. Elija Archivo > Nuevo proyecto en el menú principal.
2. Seleccione Biblioteca de clases Java de la categoría Java y haga clic en Siguiente.
3. Escriba **JUnit-Sample** para el proyecto y establezca la ubicación del proyecto.
4. Anule la selección de la opción Usar carpeta dedicada, si está seleccionada.

Para este tutorial, hay pocas razones para copiar bibliotecas de proyectos en una carpeta dedicada porque no necesitará compartir bibliotecas con otros usuarios o proyectos.

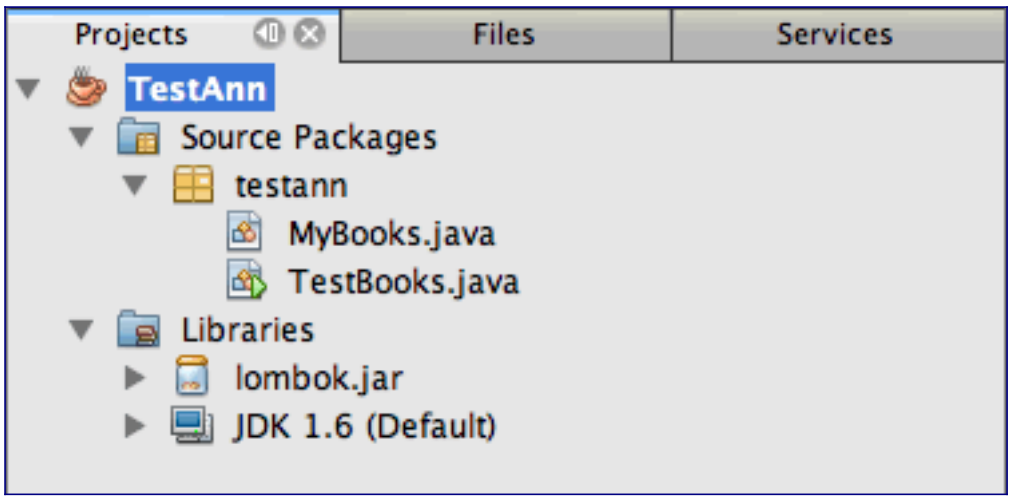
Haga clic en Finalizar.

La primera vez que crea una prueba JUnit, el IDE le solicita que seleccione una versión y luego agrega un nodo Bibliotecas de prueba y la biblioteca JUnit.

Crear las clases de Java

En este ejercicio, copiará los archivos Utils.javay Vectors.javadesde el proyecto de muestra JUnitSampleSol al proyecto de biblioteca de clases que creó.

1. En la ventana Proyectos, haga clic con el botón derecho en el nodo Paquetes de origen del proyecto **JUnit-Sample** y seleccione Nuevo > Paquete Java en el menú emergente.
2. Escriba **sample** como nombre del paquete. Haga clic en Finalizar.
3. Abra el proyecto **JUnitSampleSol** (si aún no está abierto) y expanda el nodo Paquetes de origen en la ventana Proyectos.



1. Copie las clases Utils.javay Vectors.java en el proyecto JUnitSampleSol y péguelas en el **sample** paquete fuente en JUnit-Sample.

Si observa el código fuente de las clases, puede ver que Utils.javatiene tres métodos (computeFactorial, concatWordsy normalizeWord) y que Vectors.javatiene dos métodos (equaly scalarMultiplication). El siguiente paso es crear clases de prueba para cada clase y escribir algunos casos de prueba para los métodos.

Nota Puede cerrar el proyecto JUnitSampleSol porque no lo necesitará de nuevo. El proyecto JUnitSampleSol contiene todas las pruebas descritas en este documento.

Escritura de pruebas unitarias JUnit 3

En esta parte del tutorial, creará pruebas unitarias básicas de JUnit 3 para las clases Vectors.java y Utils.java. Utilizará el IDE para crear clases de prueba de esqueleto que se basan en las clases de su proyecto. Luego modificará los métodos de prueba generados y agregará nuevos métodos de prueba.

El IDE le solicita que elija una versión de JUnit la primera vez que usa el IDE para crear pruebas para usted en el proyecto. La versión que seleccione se convierte en la versión predeterminada de JUnit y el IDE generará todas las pruebas y conjuntos de pruebas posteriores para esa versión.

Creación de una clase de prueba para Vectors.java

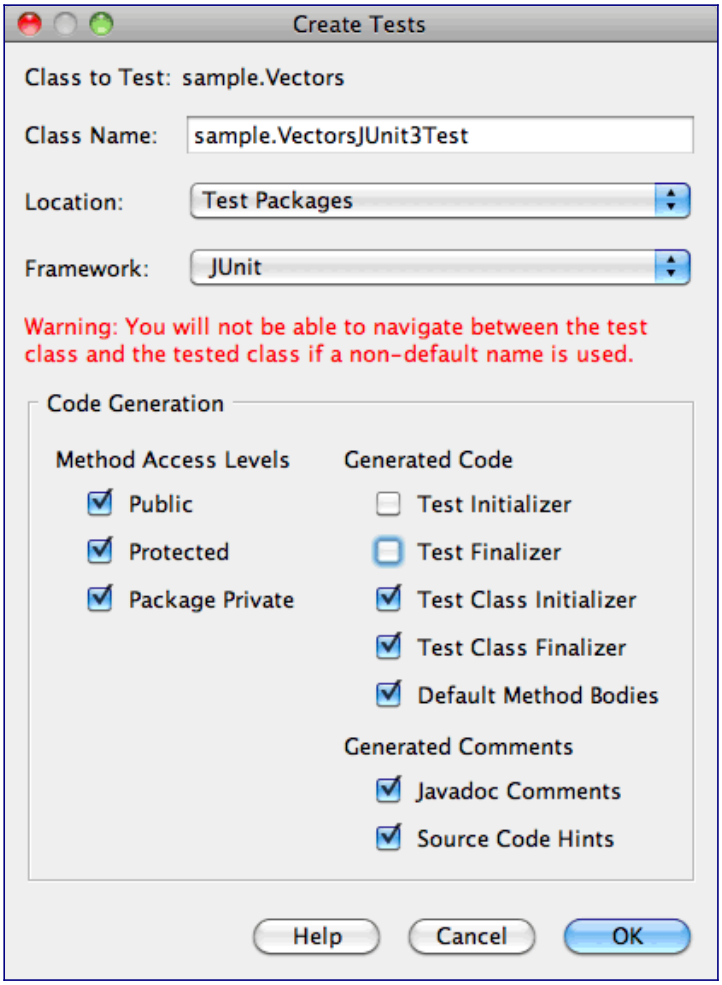
En este ejercicio, creará un esqueleto de prueba JUnit para Vectors.java. También seleccionará JUnit como marco de prueba y JUnit 3 como versión.

Nota Si está utilizando NetBeans IDE 7.1 o anterior, no necesita especificar el marco de prueba porque JUnit se especifica de forma predeterminada. Desde NetBeans IDE 7.2 en adelante, tiene la opción de especificar JUnit o TestNG como marco de prueba.

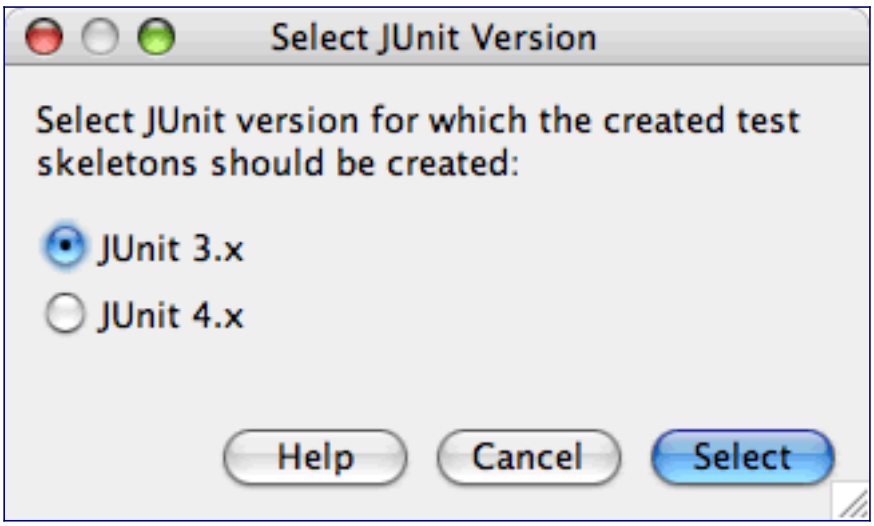
1. Haga clic con el botón derecho Vectors.java y elija Herramientas > Crear pruebas.
2. Modifique el nombre de la clase de prueba a **VectoresJUnit3Test** en el cuadro de diálogo **Crear pruebas.**

Cuando cambie el nombre de la clase de prueba, verá una advertencia sobre cómo cambiar el nombre. El nombre predeterminado se basa en el nombre de la clase que está probando, con la palabra Prueba añadida al nombre. Por ejemplo, para la clase MyClass.java, el nombre predeterminado de la clase de prueba es MyClassTest.java. Por lo general, es mejor mantener el nombre predeterminado, pero para este tutorial cambiará el nombre porque también creará pruebas JUnit 4 en el mismo paquete y los nombres de las clases de prueba deben ser únicos.

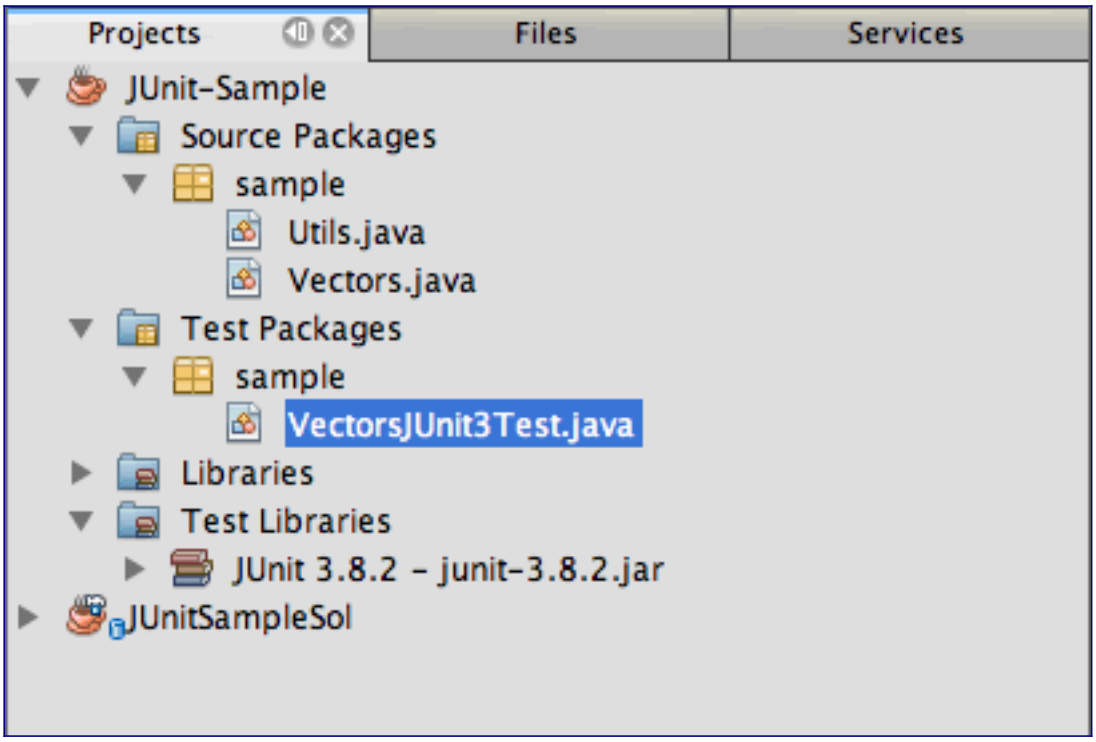
1. Seleccione JUnit en la lista desplegable Marco.
2. Anule la selección de Test Initializer y Test Finalizer. Haga clic en Aceptar.



1. Seleccione JUnit 3.x en el cuadro de diálogo Seleccionar versión de JUnit.



Cuando selecciona JUnit 3.x, el IDE agrega la biblioteca JUnit 3 al proyecto. Cuando hace clic en Seleccionar, el IDE crea la clase de prueba VectoresJUnit3Test.java en el paquete de muestra en el nodo Test Packages en la ventana Proyectos.



Un proyecto requiere un directorio para paquetes de prueba para crear pruebas. La ubicación predeterminada para el directorio de paquetes de prueba está en el nivel raíz del proyecto, pero según el tipo de proyecto, puede especificar una ubicación diferente para el directorio en el cuadro de diálogo Propiedades del proyecto.

Si observa la clase de prueba generada VectorsJUnit3Test.java en el editor, puede ver que el IDE generó la siguiente clase de prueba con métodos de prueba para los métodos equal y scalarMultiplication.

```
public class VectorsJUnit3Test extends TestCase {
    /**
     * Test of equal method, of class Vectors.
     */
    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.");
    }
    /**
     * Test of scalarMultiplication method, of class Vectors.
     */
    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.");
    }
}
```

El cuerpo del método de cada prueba generada se proporciona únicamente como guía y debe modificarse para que sea un caso de prueba real. Puede anular la selección de Cuerpos de métodos predeterminados en el cuadro de diálogo Crear pruebas si no desea que se genere el código por usted.

Cuando el IDE genera los nombres para los métodos de prueba, cada nombre de método se antepone con prueba porque JUnit 3 usa convenciones de nomenclatura y reflexión para

identificar las pruebas. Para identificar los métodos de prueba, se requiere que cada método de prueba siga la sintaxis test_<NAME>_ .

Note En JUnit 4, ya no es necesario usar esta sintaxis de nomenclatura de métodos de prueba porque puede usar anotaciones para identificar métodos de prueba y ya no se requiere que la clase de prueba se extienda TestCase.

Escribir métodos de prueba para Vectors.java

En este ejercicio, modificará los métodos de prueba generados para convertirlos en pruebas de funcionamiento y modificará los mensajes de salida predeterminados. No es necesario que modifique los mensajes de salida para ejecutar las pruebas, pero es posible que desee modificar la salida para ayudar a identificar los resultados que se muestran en la ventana de salida Resultados de la prueba JUnit.

1. Abrir VectorsJUnit3Test.java en el editor.
2. Modifique el esqueleto de prueba testScalarMultiplication cambiando el valor de println y eliminando las variables generadas. El método de prueba ahora debería tener el siguiente aspecto :

```
public  void testScalarMultiplication()
{
   System.out.println("* VectorsJUnit3Test: testScalarMultiplication()");
}
```

1. Ahora agregue algunas afirmaciones para probar el método.

```
public  void testScalarMultiplication() {
  System.out.println("* VectorsJUnit3Test: testScalarMultiplication()");
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}));
}
```

Este método de prueba utiliza el método JUnit assertEquals. Para usar la aserción, proporciona las variables de entrada y el resultado esperado. Para pasar la prueba, el método de prueba debe devolver correctamente todos los resultados esperados en función de las variables proporcionadas al ejecutar el método probado. Debe agregar un número suficiente de aserciones para cubrir las diversas permutaciones posibles.

1. Modifique el esqueleto de prueba testEqual eliminando los cuerpos de método generados y agregando lo siguiente println.

System.out.println("* VectorsJUnit3Test: testEqual()");

El método de prueba ahora debería tener el siguiente aspecto:

```
public  void testEqual() {
    System.out.println("* VectorsJUnit3Test: testEqual()");
}
```

1. Modifique el testEqual método agregando las siguientes afirmaciones (que se muestran en negrita).

```
public  void testEqual() {
    System.out.println("* VectorsJUnit3Test: testEqual()");
     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}));
}
```

Esta prueba utiliza JUnit assertTrue y assertFalsemétodos para probar una variedad de resultados posibles. Para que pase la prueba de este método, assertTruetodos deben ser verdaderos y assertFalsetodos deben ser falsos.

Guarde sus cambios.

Creación de una clase de prueba para Utils.java

Ahora crea los esqueletos de prueba para Utils.java. Cuando creó la prueba en el ejercicio anterior, el IDE le solicitó la versión de JUnit. No se le pedirá que seleccione una versión esta vez.

1. Haga clic con el botón derecho Utils.java y elija Herramientas > Crear pruebas.
2. Seleccione JUnit en la lista desplegable Marco si no está seleccionado.
3. Seleccione Test Initializer y Test Finalizer en el cuadro de diálogo, si no está seleccionado.
4. Modifique el nombre de la clase de prueba a **UtilsJUnit3Test** en el cuadro de diálogo Crear pruebas. Haga clic en Aceptar.

Cuando hace clic en Aceptar, el IDE crea el archivo de prueba UtilsJUnit3Test.java en el directorio Test Packages > **samples**. Puede ver que además de crear los esqueletos testComputeFactorialde prueba testConcatWords, y testNormalizeWord para los métodos en Utils.java, el IDE también crea el método inicializador de prueba setUp y el método finalizador de prueba tearDown.

Escribir métodos de prueba para Utils.java

En este ejercicio, agregará algunos casos de prueba que ilustran algunos elementos de prueba comunes de JUnit. También agrega println a los métodos porque algunos métodos no imprimen ningún resultado de forma predeterminada. Al agregar println a los métodos, puede mirar más tarde en la ventana de resultados de la prueba JUnit para ver si se ejecutaron los métodos y el orden en que se ejecutaro**n.**

Probar inicializadores y finalizadores

Los métodos setUp y tearDown se utilizan para inicializar y finalizar las condiciones de prueba. No necesita los métodos setUp y tearDown para probar Utils.java, pero se incluyen aquí para demostrar cómo funcionan.

El setUp método es un método de inicialización de prueba y se ejecuta 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, use el método de inicialización de prueba.

El tearDown método es un método de finalización de prueba y se ejecuta 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 cualquier dato que se requiera al ejecutar los casos de prueba.

Prueba usando una aserción simple

Este caso de prueba simple prueba el concatWords método. En lugar de usar el método de prueba generado testConcatWords, agregará un nuevo método de prueba llamado testHelloWorld que usa una sola afirmación simple para probar si el método concatena las cadenas correctamente. El assertEquals en el caso de prueba usa la sintaxis para probar si el resultado esperado es igual al resultado real. En este caso, si la entrada al método es " ", " ", " " y " ", el resultado esperado debería ser igual a .assertEquals(*EXPECTED_RESULT, ACTUAL_RESULT*)concatWordsHello, world!"Hello, world!"

1. Agregue una println declaración para mostrar texto sobre la prueba en la ventana Resultados de la prueba JUnit.

```
public  void testHelloWorld() {
System.out.println("* UtilsJUnit3Test: test method 1 - testHelloWorld()");
 assertEquals("Hello, world!", Utils.concatWords("Hello", ", ", "world", "!"));
```

Probar usando un tiempo de espera

Esta prueba demuestra cómo comprobar si un método tarda demasiado en completarse. Si el método tarda demasiado, el subproceso de prueba se interrumpe y la prueba falla. Puede especificar el límite de tiempo en la prueba.

El método de prueba invoca el computeFactorial método en Utils.java. Puede suponer que el computeFactorial método es correcto, pero en este caso desea probar si el cálculo se completa en 1000 milisegundos. El computeFactorial subproceso y un subproceso de prueba se inician al mismo tiempo. El subproceso de prueba se detendrá después de 1000 milisegundos y lanzará un mensaje a TimeoutExceptionmenos que el computeFactorial subproceso se complete primero. Agregará un mensaje para que se muestre un mensaje si TimeoutException se lanza un.

1. Elimine el método de prueba generado testComputeFactorial.
2. Agrega el testWithTimeout método que calcula el factorial de un número generado aleatoriamente.

```
*public void testWithTimeout() throws InterruptedException, TimeoutException { final int factorialOf = 1 + (int) (30000 * Math.random());
System.out.println("informática" + factorialOf + '!');
Thread testThread = new Thread() {
public void run() {
  System.out.println(factorialOf + "! = " + Utils.computeFactorial(factorialOf));
   }
     };
   }*
```

1. Arregle sus importaciones para importar java.util.concurrent.TimeoutException.
2. Agregue el siguiente código (que se muestra en negrita) al método para interrumpir el hilo y mostrar un mensaje si la prueba tarda demasiado en ejecutarse.

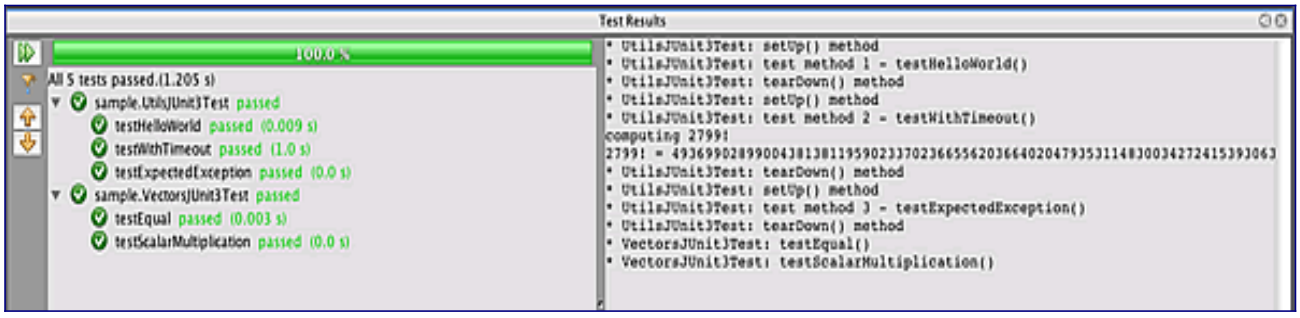
```
 testThread.start();
Thread.sleep(10000000);
testThread.interrupt();


if (testThread.isInterrupted()) {
   throw new TimeoutException("the test took too long to complete");
}
```

Ejecución de las pruebas

Cuando ejecuta una prueba JUnit, los resultados se muestran en la ventana Resultados de la prueba del IDE. Puede ejecutar clases de prueba JUnit individuales o puede elegir Ejecutar > Probar *PROJECT_NAME* en el menú principal para ejecutar todas las pruebas del proyecto. Si elige Ejecutar > Probar, el IDE ejecuta todas las clases de prueba en la carpeta Paquetes de prueba. Para ejecutar una clase de prueba individual, haga clic con el botón derecho en la clase de prueba en el nodo Paquetes de prueba y elija Ejecutar archivo.

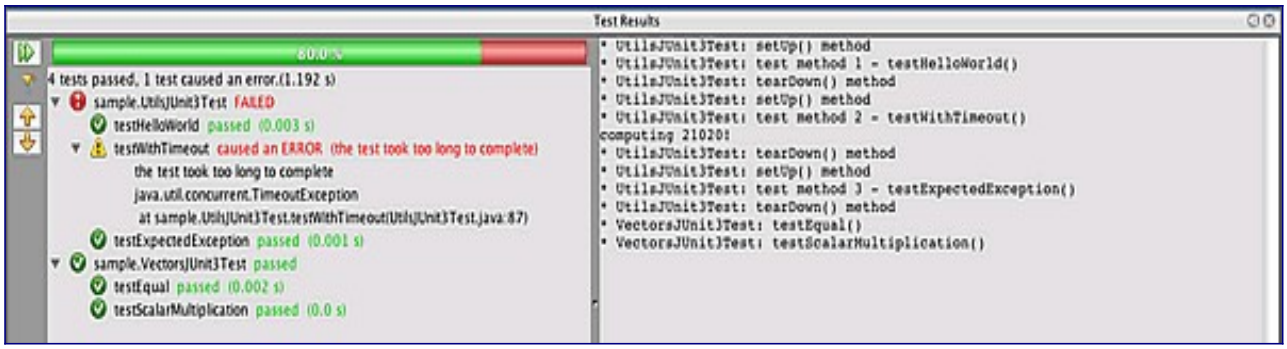
1. Elija Run > Set Main Project en el menú principal y seleccione el proyecto JUnit-Sample.
2. Elija Ejecutar > Proyecto de prueba (JUnit-Sample) en el menú principal.
3. Elija Ventana > Herramientas IDE > Resultados de la prueba para abrir la ventana Resultados de la prueba.

Cuando ejecute la prueba, verá uno de los siguientes resultados en la ventana Resultados de la prueba JUnit.



En esta imagen (haga clic en la imagen para ver una imagen más grande) puede ver que el proyecto pasó todas las pruebas. El panel izquierdo muestra los resultados de los métodos de prueba individuales y el panel derecho muestra el resultado de la prueba. Si observa la salida, puede ver el orden en que se ejecutaron las pruebas. El println que agregó a cada uno de los métodos de prueba imprimió el nombre de la prueba en la ventana de salida. También puede ver que UtilJUnit3Testel setUp método se ejecutó antes de cada método de prueba y que el

tearDown método se ejecutó después de cada método.



En esta imagen (haga clic en la imagen para ver una imagen más grande) puede ver que el proyecto falló una de las pruebas. El testTimeout método tardó demasiado en completarse y el subproceso de prueba se interrumpió, lo que provocó que la prueba fallara. Llevó más de 1000 milisegundos calcular el factorial del número generado aleatoriamente (22991).

El siguiente paso después de crear sus clases de prueba de unidad es crear suites de prueba. Consulte para ver cómo ejecutar pruebas específicas como un grupo para que no tenga que ejecutar cada prueba individualmente.

Escritura de pruebas JUnit 4

En este ejercicio, creará pruebas unitarias JUnit 4 para las clases Vectors.java y Utils.java. Los casos de prueba JUnit 4 son los mismos que los casos de prueba JUnit 3, pero verá que la sintaxis para escribir las pruebas es más simple.

Utilizará los asistentes del IDE para crear esqueletos de prueba basados en las clases de su proyecto. La primera vez que usa el IDE para crear algunos esqueletos de prueba para usted, el IDE le pide que elija la versión JUnit.

Nota. Si ya seleccionó JUnit 3.x como la versión predeterminada para sus pruebas, debe cambiar la versión predeterminada a JUnit 4.x. Para cambiar la versión predeterminada de JUnit, expanda el nodo Bibliotecas de prueba, haga clic con el botón derecho en la biblioteca JUnit y elija Eliminar. Ahora puede usar el cuadro de diálogo Agregar biblioteca para agregar explícitamente la biblioteca JUnit 4 o puede seleccionar la versión 4.x cuando se le solicite que seleccione la versión JUnit cuando cree una nueva prueba. Todavía puede ejecutar pruebas JUnit 3, pero cualquier prueba nueva que cree utilizará JUnit 4.

Creación de una clase de prueba para Vectors.java

En este ejercicio, creará los esqueletos de prueba JUnit para Vectors.java.

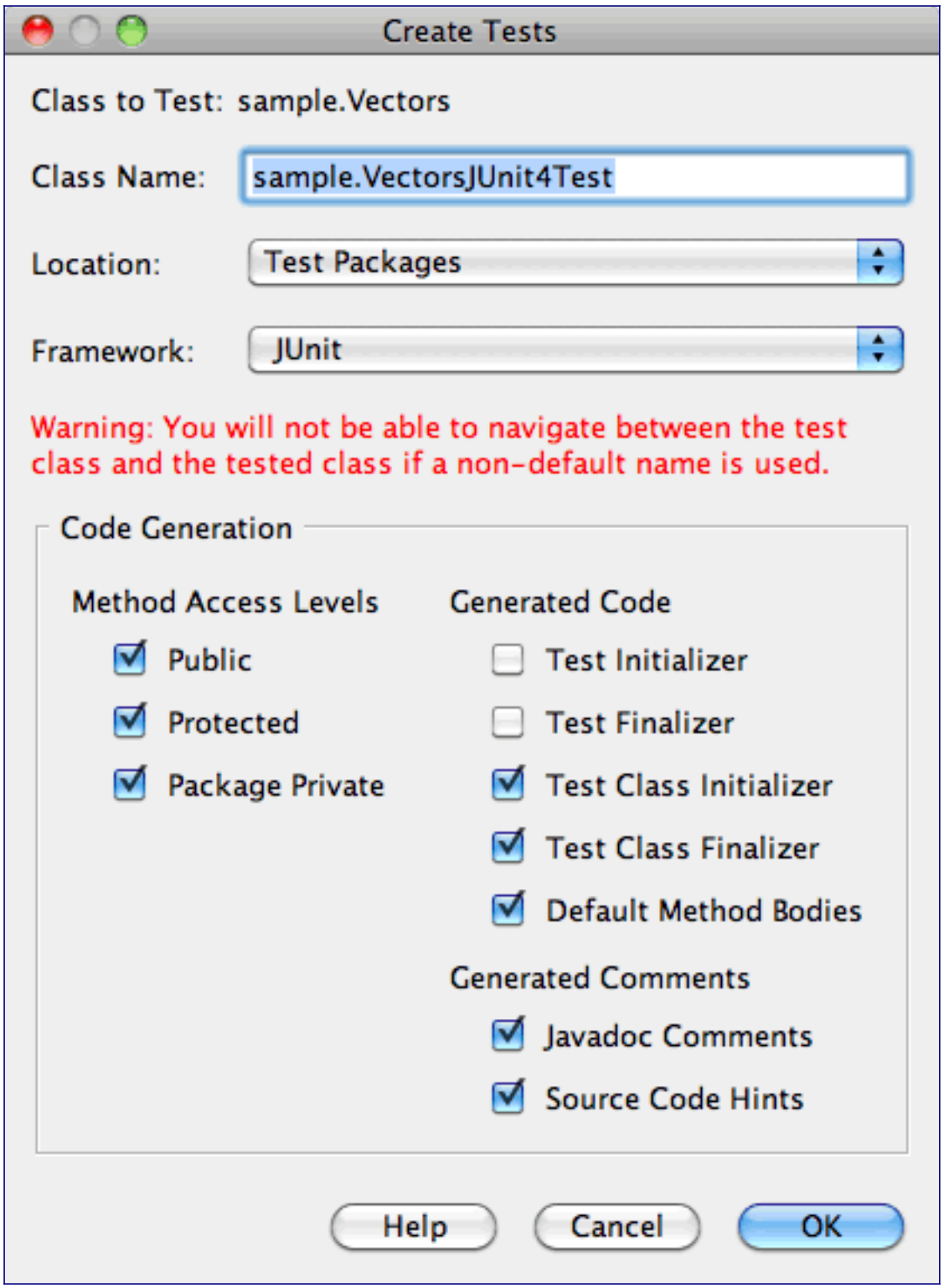
Nota

Si está utilizando NetBeans IDE 7.1 o anterior, no necesita especificar el marco de prueba porque JUnit se especifica de forma predeterminada. Desde NetBeans IDE 7.2 en adelante, tiene la opción de especificar JUnit o TestNG como marco de prueba.

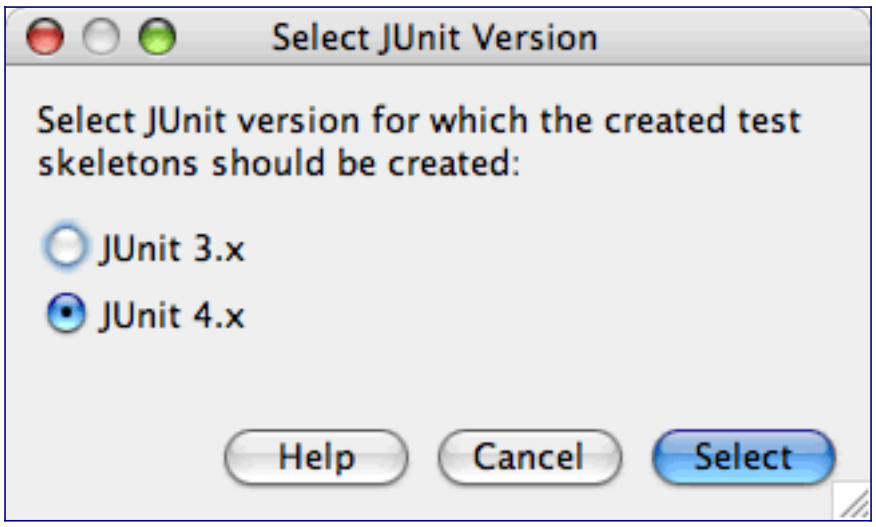
1. Haga clic con el botón derecho Vectors.java y elija Herramientas > Crear pruebas.
2. Modifique el nombre de la clase de prueba a **VectoresJUnit4Test** en el cuadro de diálogo Crear pruebas.

Cuando cambie el nombre de la clase de prueba, verá una advertencia sobre cómo cambiar el nombre. El nombre predeterminado se basa en el nombre de la clase que está probando, con la palabra Prueba añadida al nombre. Por ejemplo, para la clase MyClass.java, el nombre predeterminado de la clase de prueba es MyClassTest.java. A diferencia de JUnit 3, en JUnit 4, no se requiere que la prueba termine con la palabra Prueba. Por lo general, es mejor mantener el nombre predeterminado, pero debido a que está creando todas las pruebas JUnit en el mismo paquete en este tutorial, los nombres de las clases de prueba deben ser únicos.

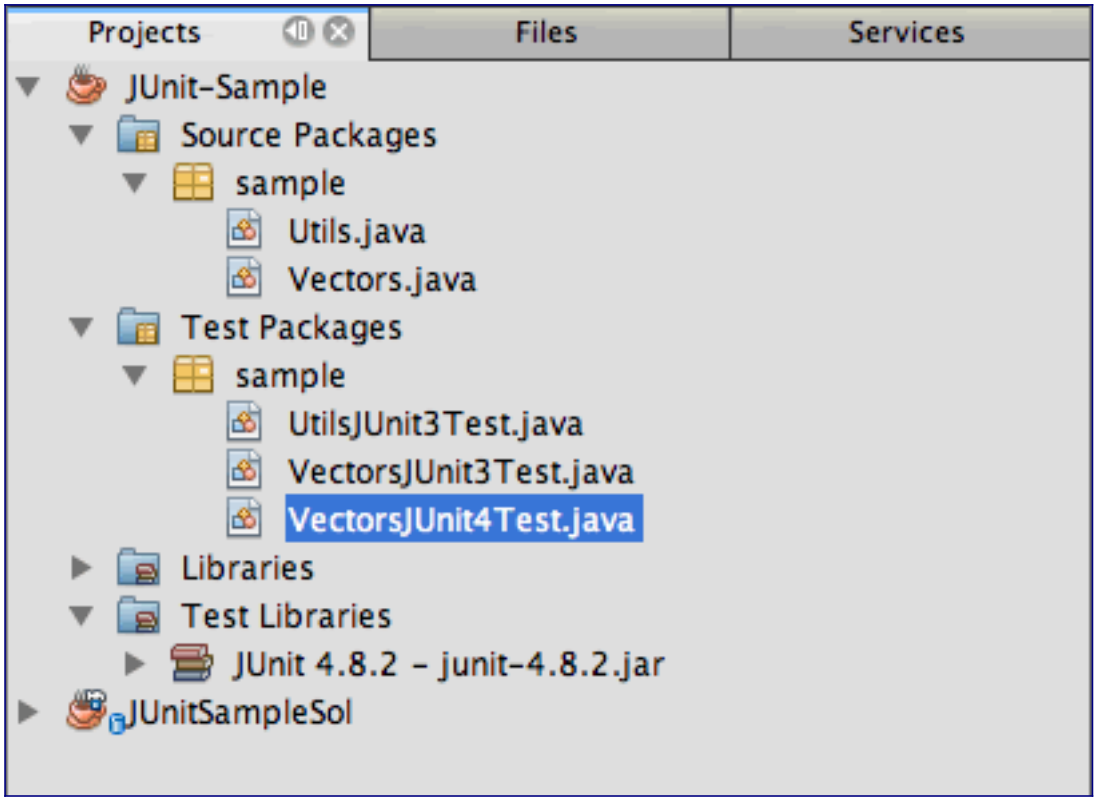
1. Seleccione JUnit en la lista desplegable Marco.
2. Anule la selección de Test Initializer y Test Finalizer. Haga clic en Aceptar.



1. Seleccione JUnit 4.x en el cuadro de diálogo Seleccionar versión de JUnit. Haga clic en Seleccionar.



Cuando hace clic en Aceptar, el IDE crea la VectorsJUnit4Test.java clase de prueba en el sample paquete en el nodo Paquetes de prueba en la ventana Proyectos.



Nota

Un proyecto requiere un directorio para paquetes de prueba para crear pruebas. La ubicación predeterminada para el directorio de paquetes de prueba está en el nivel raíz del proyecto, pero puede especificar una ubicación diferente para el directorio en el cuadro de diálogo Propiedades del proyecto.

Si mira VectorsJUnit3Test.javaen el editor, puede ver que el IDE generó los métodos de prueba testEqual y testScalarMultiplication. En VectorsJUnit4Test.java, cada método de prueba se

anota con @Test. El IDE generó los nombres para los métodos de prueba en función de los nombres del método, Vectors.java pero no es necesario que el nombre del método de prueba se test anteponga. El cuerpo predeterminado de cada método de prueba generado se proporciona únicamente como guía y debe modificarse para que sean casos de prueba reales.

Puede anular la selección de Cuerpos de método predeterminados en el cuadro de diálogo Crear pruebas si no desea que se generen automáticamente los cuerpos del método.

El IDE también generó los siguientes métodos de inicialización y finalización de clase de prueba:

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

El IDE genera los métodos de inicialización y finalización de clase de forma predeterminada al crear clases de prueba JUnit 4. Las anotaciones @BeforeClassy @AfterClassse utilizan para marcar métodos que deben ejecutarse antes y después de ejecutar la clase de prueba. Puede eliminar los métodos porque no los necesitará para probar Vectors.java.

Puede configurar los métodos que se generan por defecto configurando las opciones de JUnit en la ventana Opciones.

Nota Para las pruebas de JUnit 4, tenga en cuenta que, de forma predeterminada, el IDE agrega una declaración de importación estática para org.junit.Assert.*.

Escribir métodos de prueba para Vectors.java

En este ejercicio modificará cada uno de los métodos de prueba generados para probar los métodos utilizando el assert método JUnit y para cambiar los nombres de los métodos de prueba. En JUnit 4, tiene mayor flexibilidad al nombrar métodos de prueba porque los métodos de prueba se indican mediante la @Testanotación y no requieren la palabra test antepuesta a los nombres de los métodos de prueba.

1. Abrir VectorsJUnit4Test.javaen el editor.
2. Modifique el método de prueba testScalarMultiplication cambiando el nombre del método, el valor de println y eliminando las variables generadas. El método de prueba ahora debería tener el siguiente aspecto (los cambios se muestran en negrita):

```
@Test
public  void  * ScalarMultiplicationCheck * () {
```

```
    System.out.println( " ** VectoresJUnit4Test: ScalarMultiplicationCheck()* " );
   afirmarEquals(expResult, resultado);
}
```

Nota: Al escribir pruebas, no es necesario cambiar la salida impresa. Haga esto en este ejercicio para que sea más fácil identificar los resultados de la prueba en la ventana de resultados.

1. Ahora agregue algunas afirmaciones para probar el método.

```
@Test
public  void ScalarMultiplicationCheck() {
    System . out .println( " * VectoresJUnit4Test: ScalarMultiplicationCheck() " );
   * assertEquals(   0 , Vectores . escalarMultiplication( new  int [] { 0 , 0 }, new  int [] { 0 , 0 }));
   afirmarEquals( 39 , Vectores . escalarMultiplicación( new  int [] { 3 , 4 }, new  int [] { 5 , 6 }));
   afirmarEquals( - 39 , Vectores . escalarMultiplicación( new  int [] { - 3 , 4 }, new  int [] { 5 , - 6 }));
   assertEquals(   0 , Vectores . escalarMultiplication( new  int [] { 5 , 9 }, new  int [] { - 9 , 5 }));
   assertEquals( 100 , Vectores . escalarMultiplicación( new  int [] { 6 , 8 }, new  int [] { 6 , 8 })); *
}
```

En este método de prueba, utiliza el método JUnit assertEquals. Para usar la aserción, proporciona las variables de entrada y el resultado esperado. Para pasar la prueba, el método de prueba debe devolver correctamente todos los resultados esperados en función de las variables proporcionadas al ejecutar el método probado. Debe agregar un número suficiente de aserciones para cubrir las diversas permutaciones posibles.

1. Cambie el nombre del testEqual método de prueba a equalsCheck.
2. Elimine el cuerpo del método generado del método de equalsCheckprueba.
3. Agregue lo siguiente printlnal equalsCheck método de prueba. **System.out.println("** VectoresJUnit4Test: equalsCheck()");*

El método de prueba ahora debería tener el siguiente aspecto:

```
@Prueba
public  void equalsCheck() {
    System . out .println( " *VectoresJUnit4Test:EqualsCheck() " );
}
```

1. Modifique el equalsCheckmétodo agregando las siguientes afirmaciones (que se muestran en negrita).

```
@Prueba
public  void equalsCheck() {
    System . out .println( " *VectoresJUnit4Test:EqualsCheck() " );
   * assertTrue( Vectores . equal( new  int [] {}, new  int [] {}));
   afirmarVerdadero( Vectores . equal( new  int [] { 0 }, new  int [] { 0 }));
   afirmarVerdadero( Vectores . equal( new  int [] { 0 , 0 }, new  int [] { 0 , 0 }));
   afirmarVerdadero( Vectores . equal( new  int [] { 0 , 0 , 0 }, new  int [] { 0 , 0 , 0 }));
   afirmarVerdadero( Vectores . equal( new  int [] { 5 , 6 , 7 }, new  int [] { 5 , 6 , 7 }));

   afirmarFalse( Vectores . igual( new  int [] {}, new  int [] { 0 }));
   afirmarFalse( Vectores . igual( new  int [] { 0 }, new  int [] { 0 , 0 }));
   afirmarFalse( Vectores . equal( new  int [] { 0 , 0 }, new  int [] { 0 , 0 , 0 }));
   afirmarFalse( Vectores . equal( new  int [] { 0 , 0 , 0 }, new  int [] { 0 , 0 }));
   afirmarFalse( Vectores . equal( new  int [] { 0 , 0 }, new  int [] { 0 }));
   afirmarFalse( Vectores . igual( new  int [] { 0 }, new  int [] {}));

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

Esta prueba utiliza JUnit assertTruey assertFalse métodos para probar una variedad de resultados posibles. Para que pase la prueba de este método, assertTrue todos deben ser verdaderos y assertFalse todos deben ser falsos.

Creación de una clase de prueba paraUtils.java

Ahora creará los métodos de prueba JUnit para Utils.java. Cuando creó la clase de prueba en el ejercicio anterior, el IDE le solicitó la versión de JUnit. No se le solicita que seleccione una versión esta vez porque ya seleccionó la versión JUnit y todas las pruebas JUnit posteriores se crean en esa versión.

Nota — Todavía puede escribir y ejecutar pruebas JUnit 3 si selecciona JUnit 4 como la versión, pero el IDE usa la plantilla JUnit 4 para generar esqueletos de prueba.

1. Haga clic con el botón derecho Utils.java y elija Herramientas > Crear pruebas.
2. Seleccione JUnit en la lista desplegable Marco si no está seleccionado.
3. Seleccione Test Initializer y Test Finalizer en el cuadro de diálogo si no está seleccionado.
4. Modifique el nombre de la clase de prueba a **UtilsJUnit4Test** en el cuadro de diálogo Crear pruebas. Haga clic en Aceptar.

Cuando hace clic en Aceptar, el IDE crea el archivo de prueba UtilsJUnit4Test.javaen el directorio Test Packages > sample. Puede ver que el IDE generó los métodos testComputeFactorialde prueba testConcatWords, y testNormalizeWordpara los métodos en Utils.java. El IDE también generó métodos de inicialización y finalización para la prueba y la clase de prueba.

Escribir métodos de prueba paraUtils.java

En este ejercicio agregará casos de prueba que ilustran algunos elementos de prueba comunes de JUnit. También agregará un println a los métodos porque algunos métodos no imprimen ningún resultado en la ventana Resultados de la prueba JUnit para indicar que se ejecutaron o para indicar que el método pasó la prueba. Al agregar un println a los métodos, puede ver si los métodos se ejecutaron y el orden en que se ejecutaron..

Inicializadores y finalizadores de prueba

Cuando creó la clase de prueba para Utils.java, el IDE generó métodos de inicializador y finalizador anotados. Puede elegir cualquier nombre para el nombre del método porque no se requiere ninguna convención de nomenclatura.

Nota	No necesita los métodos inicializador y finalizador para probar Utils.java, pero se incluyen en este tutorial para demostrar cómo funcionan.

En JUnit 4 puede usar anotaciones para marcar los siguientes tipos de métodos de inicializador y finalizador.

- **Test Class Initializer.** @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 demás 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. Luego podría cerrar la conexión con el finalizador de clase de prueba.

- **Finalizador de clase de prueba.** La @AfterClassanotación marca un método como método finalizador de clase de prueba. Un método de 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.
- **Inicializador de prueba.** La @Beforeanotación 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 @Afteranotación 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 las pruebas, pero es posible que necesite un finalizador para limpiar cualquier

dato que se requiera al ejecutar los casos de prueba.

Realice los siguientes cambios (que se muestran en negrita) en UtilsJUnit4Test.java.

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

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

@Antes de la configuración de
void público  () {
   * System . out .println( " * UtilsJUnit4Test: @Antes del método " ); *
}

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

Cuando ejecuta la clase de prueba, el printlntexto que agregó se muestra en el panel de salida de la ventana Resultados de la prueba JUnit. Si no agrega el println, no hay ningún resultado que indique que se ejecutaron los métodos de inicialización y finalización.

Prueba usando una aserción simple

Este caso de prueba simple prueba el concatWords método. En lugar de usar el método de prueba generado testConcatWords, agregará un nuevo método de prueba llamado helloWorldCheckque usa una sola afirmación simple para probar si el método concatena las cadenas correctamente. El assertEqualsen el caso de prueba usa la sintaxis para probar si el resultado esperado es igual al resultado real. En este caso, si la entrada al método es " ", " ", " " y " ", el resultado esperado debería ser igual a .assertEquals(*EXPECTED_RESULT, ACTUAL_RESULT*)concatWordsHello,world!"Hello, world!"

1. Elimine el método de prueba generado testConcatWords.
2. Agregue el siguiente helloWorldCheckmétodo para probar Utils.concatWords.

@Test public void helloWorldCheck()

{ assertEquals("¡Hola, mundo!", Utils.concatWords("Hola", ", ", "mundo", "!")); }

1. Agregue una ***println*** declaración para mostrar texto sobre la prueba en la ventana Resultados de la prueba JUnit.

```
@Prueba
public  void helloWorldCheck() {
   System . out .println( " * UtilsJUnit4Test: método de prueba 1 - helloWorldCheck() " );
  assertEquals( " ¡Hola, mundo! " , Utils . concatWords( " Hola " , " , " , " mundo " , " ! " ));
```

Probar usando un tiempo de espera

Esta prueba demuestra cómo comprobar si un método tarda demasiado en completarse. Si el método tarda demasiado, el subproceso de prueba se interrumpe y la prueba falla. Puede especificar el límite de tiempo en la prueba.

El método de prueba invoca el computeFactorialmétodo en Utils.java. Puede suponer que el computeFactorialmétodo es correcto, pero en este caso desea probar si el cálculo se completa en 1000 milisegundos. Para ello, interrumpa el subproceso de prueba después de 1000 milisegundos. Si el subproceso se interrumpe, el método de prueba arroja un archivo TimeoutException.

1. Elimine el método de prueba generado testComputeFactorial.
2. Agregue el testWithTimeoutmétodo que calcula el factorial de un número generado aleatoriamente.

   @Test public void testWithTimeout()

   { final int factorialOf = 1 + (int) (30000 * Math.random()); System.out.println("informática" + factorialOf + '!');

   System.out.println(factorialOf + "! = " + Utils.computeFactorial(factorialOf)); }

   1. Agregue el siguiente código (que se muestra en negrita) para establecer el tiempo de espera y para interrumpir el subproceso si el método tarda demasiado en ejecutarse.

```
@Test* ( timeout = 1000 ) *
public  void testWithTimeout() {
   final  int factorialOf =  1  + ( int ) ( 30000  *  Math . random());
```

Puede ver que el tiempo de espera está establecido en 1000 milisegundos.

1. Agregue lo siguiente println(que se muestra en negrita) para imprimir el texto sobre la prueba en la ventana Resultados de la prueba JUnit.

```
@Test ( tiempo de espera = 1000 )
 public  void testWithTimeout() {
    * System . out .println( " * UtilsJUnit4Test: método de prueba 2 - testWithTimeout() " ); *
   int final  factorialOf = 1 + ( int ) ( 30000 * Math . random());
   System.out.println( " computación " + factorialOf + ' !     ' );
```

Prueba de una excepción esperada

Esta prueba demuestra cómo probar una excepción esperada. El método falla si no arroja la excepción esperada especificada. En este caso, está probando que el computeFactorialmétodo arroja un IllegalArgumentExceptionsi la variable de entrada es un número negativo (-5).

1. Agregue el siguiente testExpectedExceptionmétodo que invoca el computeFactorialmétodo con una entrada de -5.

   @Test public void checkExpectedException()

   { final int factorialOf = -5;

   System.out.println(factorialOf + "! = " + Utils.computeFactorial(factorialOf)); }

   1. Agregue la siguiente propiedad (mostrada en negrita) a la @Testanotación para especificar que se espera que la prueba genere IllegalArgumentException.

```
@Test* ( esperado = IllegalArgumentException . clase) *
public  void checkExpectedException() {
    final  int factorialOf =  - 5 ;
   System.out.println(factorialOf +  " ! = "  +  Utils . computeFactorial(factorialOf));
}
```

   1. Agregue lo siguiente println(que se muestra en negrita) para imprimir el texto sobre la prueba en la ventana Resultados de la prueba JUnit.

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

Deshabilitar una prueba

Esta prueba demuestra cómo deshabilitar temporalmente un método de prueba. En JUnit 4,

simplemente agrega la @Ignoreanotación para deshabilitar la prueba.

1. Elimine el método de prueba generado testNormalizeWord.
2. Agregue el siguiente método de prueba a la clase de prueba.

   @Test public void temporalmenteDisabledTest() throws Exception { System.out.println(" UtilsJUnit4Test: método de prueba 4 - checkExpectedException()"); afirmarEquals("Malm\u00f6", Utils.normalizeWord("Malm\u0308")); }*

El método de prueba temporarilyDisabledTestse ejecutará si ejecuta la clase de prueba.

1. Agregue la @Ignoreanotación (mostrada en negrita) arriba @Testpara deshabilitar la prueba. **@Ignorar**

```
@Test
public  void temporalmenteDisabledTest() throws Exception {
    System . out .println( " * UtilsJUnit4Test: método de prueba 4 - checkExpectedException() " );
  afirmarEquals( " Malm \u 00f6 " , Utils . normalizeWord( " Malmö \u 0308 " ));
}
```

1. Arregle sus importaciones para importar org.junit.Ignore.

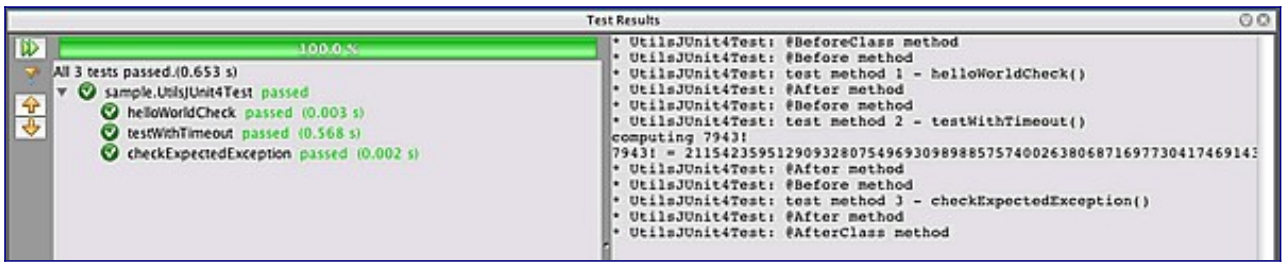
Ahora que ha escrito las pruebas, puede ejecutar la prueba y ver el resultado de la prueba en la ventana Resultados de la prueba JUnit.

Ejecución de las pruebas

Puede ejecutar pruebas JUnit en toda la aplicación o en archivos individuales y ver los resultados en el IDE. La forma más sencilla de ejecutar todas las pruebas unitarias del proyecto es elegir Ejecutar > Probar *<NOMBRE_PROYECTO>* en el menú principal. Si elige este método, el IDE ejecuta todas las clases de prueba en los paquetes de prueba. Para ejecutar una clase de prueba individual, haga clic con el botón derecho en la clase de prueba en el nodo Paquetes de prueba y elija Ejecutar archivo.

1. Haga clic derecho UtilsJUnit4Test.javaen la ventana Proyectos.
2. Elija Archivo de prueba.
3. Elija Ventana > Herramientas IDE > Resultados de la prueba para abrir la ventana Resultados de la prueba.

Cuando ejecuta UtilsJUnit4Test.javael IDE, solo ejecuta las pruebas en la clase de prueba. Si la clase pasa todas las pruebas, verá algo similar a la siguiente imagen en la ventana Resultados de la prueba JUnit.



En esta imagen (haga clic en la imagen para ver una imagen más grande) puede ver que el IDE ejecutó la prueba JUnit Utils.javay que la clase pasó todas las pruebas. El panel izquierdo muestra los resultados de los métodos de prueba individuales y el panel derecho muestra el resultado de la prueba. Si observa la salida, puede ver el orden en que se ejecutaron las pruebas. El printlnque agregó a cada uno de los métodos de prueba imprimió el nombre de la prueba en la ventana Resultados de la prueba y la ventana Salida.

Puede ver que en UtilsJUnit4Testel método de inicialización de clase de prueba anotado con @BeforeClassse ejecutó antes que cualquiera de los otros métodos y se ejecutó solo una vez. El método finalizador de clase de prueba anotado con @AfterClassse ejecutó en último lugar, después de todos los demás métodos de la clase. El método inicializador de prueba anotado con @Beforese ejecutó antes de cada método de prueba.

Los controles en el lado izquierdo de la ventana Resultados de la prueba le permiten volver a ejecutar la prueba fácilmente. Puede usar el filtro para alternar entre mostrar todos los resultados de las pruebas o solo las pruebas fallidas. Las flechas le permiten saltar al siguiente error o al error anterior.

Cuando hace clic con el botón derecho en el resultado de una prueba en la ventana Resultados de la prueba, el menú emergente le permite elegir ir a la fuente de la prueba, ejecutar la prueba nuevamente o depurar la prueba.

El siguiente paso después de crear sus clases de prueba unitaria es crear suites de prueba. Consulte en JUnit4 para ver cómo ejecutar pruebas específicas como un grupo para que no tenga que ejecutar cada prueba individualmente.

Creación de conjuntos de pruebas

Al crear pruebas para un proyecto, generalmente terminarás con muchas clases de prueba. Si bien puedes ejecutar clases de prueba individualmente o ejecutar todas las pruebas en un proyecto, en muchos casos querrás ejecutar un subconjunto de las pruebas o ejecutar pruebas en un orden específico. Puedes lograr esto creando una o más suites de pruebas. Por ejemplo, puedes crear suites de pruebas que evalúen aspectos específicos de tu código o condiciones específicas.

Una suite de pruebas es básicamente una clase con un método que invoca los casos de prueba especificados, como clases de prueba específicas, métodos de prueba en clases de prueba y otras suites de pruebas. Una suite de pruebas puede incluirse como parte de una clase de prueba, pero las mejores prácticas recomiendan crear clases de suites de pruebas individuales.

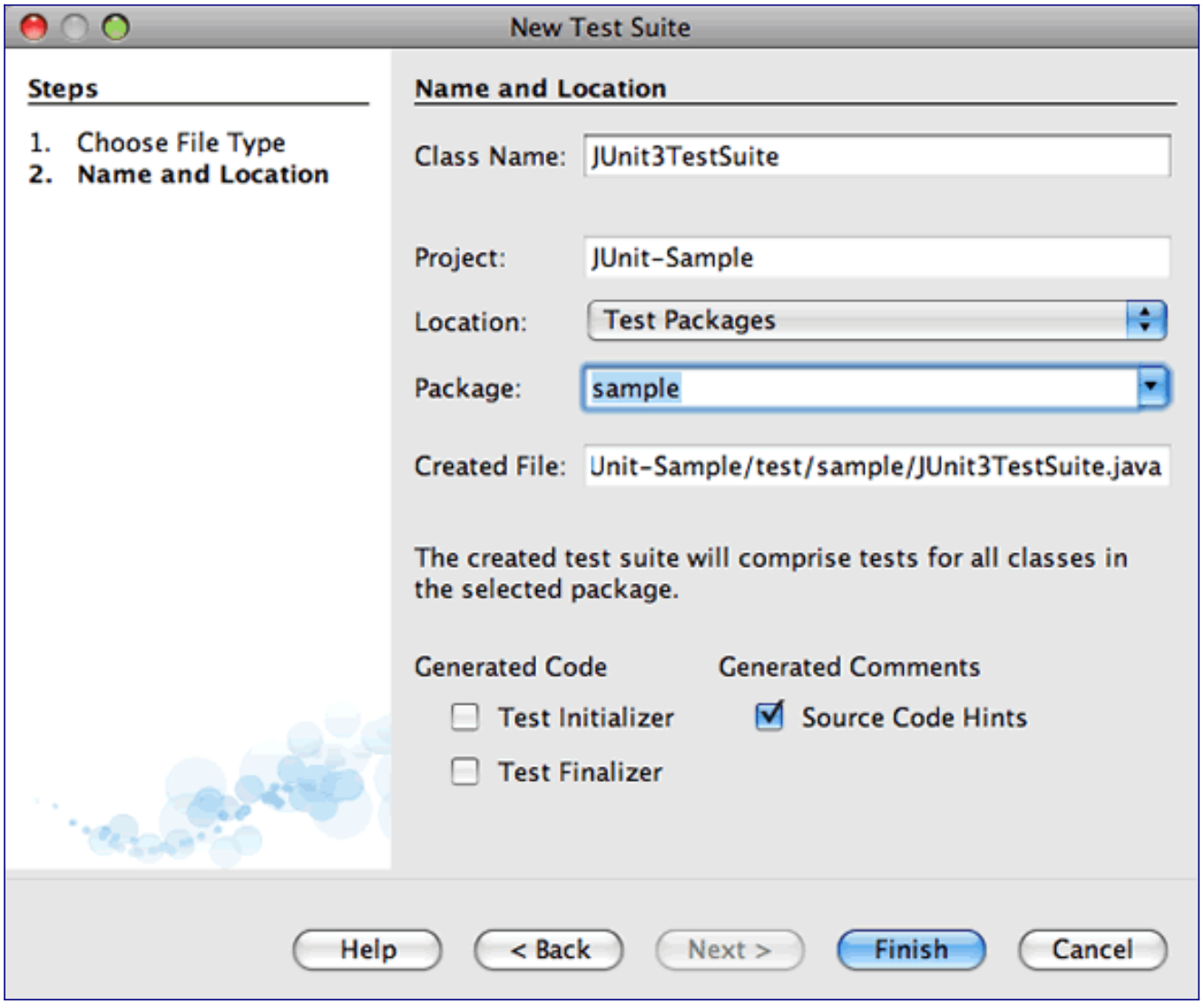
Puedes crear suites de pruebas para JUnit 3 y JUnit 4 en tu proyecto manualmente, o la IDE

puede generar las suites por ti. Cuando utilizas la IDE para generar una suite de pruebas, de forma predeterminada, la IDE genera código para invocar todas las clases de prueba en el mismo paquete que la suite de pruebas. Después de crear la suite de pruebas, puedes modificar la clase para especificar las pruebas que deseas ejecutar como parte de esa suite.

Creación de Suites de Pruebas JUnit 3

Si seleccionaste JUnit 3 como la versión para tus pruebas, la IDE puede generar suites de pruebas JUnit 3 basadas en las clases de prueba en el paquete de pruebas. En JUnit 3, especificas las clases de prueba que se incluirán en la suite de pruebas creando una instancia de `TestSuite` y utilizando el método `addTest` para cada prueba.

1. Haz clic derecho en el nodo del proyecto JUnit-Sample en la ventana de Proyectos y elige Nuevo > Otro para abrir el asistente de Nuevo Archivo.
2. Selecciona Test Suite en la categoría de Pruebas Unitarias. Haz clic en Siguiente.
3. Escribe JUnit3TestSuite para el nombre de la clase.
4. Selecciona el paquete de muestra para crear la suite de pruebas en la carpeta de muestra en el paquete de pruebas.
5. Desselecciona Inicializador de Pruebas y Finalizador de Pruebas. Haz clic en Finalizar.



Cuando hagas clic en "Finalizar", la IDE creará la clase de la suite de pruebas en el paquete de muestra y abrirá la clase en el editor. La suite de pruebas contendrá el siguiente código.

```
public class JUnit3TestSuite extends TestCase {
   public JUnit3TestSuite(String testName) {
      super(testName);
   }

   public static Test suite() {
      TestSuite suite = new TestSuite("JUnit3TestSuite");
      return suite;
   }
}
```

1. Modifica el métod ***suite()*** method añadir las clases de prueba que se ejecutarán como parte de la suite.

```
public JUnit3TestSuite(String testName) {
   super(testName);
}
public static Test suite() {
   TestSuite suite = new TestSuite("JUnit3TestSuite");
   *suite.addTest(new TestSuite(sample.VectorsJUnit3Test.class));
   suite.addTest(new TestSuite(sample.UtilsJUnit3Test.class));*
   return suite;
}
```

1. Guarda los cambios.

Creando JUnit 4 Test Suites

Si seleccionaste JUnit 4 como la versión para tus pruebas, la IDE puede generar suites de pruebas JUnit 4. JUnit 4 es compatible hacia atrás, por lo que puedes ejecutar suites de pruebas JUnit 4 que contengan pruebas de JUnit 4 y JUnit 3. En las suites de pruebas JUnit 4, especificas las clases de prueba a incluir como valores de la anotación `@Suite`.

Nota: Para ejecutar suites de pruebas JUnit 3 como parte de una suite de pruebas JUnit 4, se requiere JUnit 4.4 o superior.

1. Haz clic derecho en el nodo del proyecto en la ventana de Proyectos y elige Nuevo > Otro

para abrir el asistente de Nuevo Archivo.
2. Selecciona Test Suite en la categoría de Pruebas Unitarias. Haz clic en Siguiente.
3. Escribe JUnit4TestSuite para el nombre del archivo.
4. Selecciona el paquete de muestra para crear la suite de pruebas en la carpeta de muestra en el paquete de pruebas.
5. Desmarca Inicializador de Pruebas y Finalizador de Pruebas. Haz clic en Finalizar.

Cuando hagas clic en "Finalizar", la IDE creará la clase de la suite de pruebas en el paquete de muestra y abrirá la clase en el editor. La suite de pruebas contendrá código similar al siguiente.

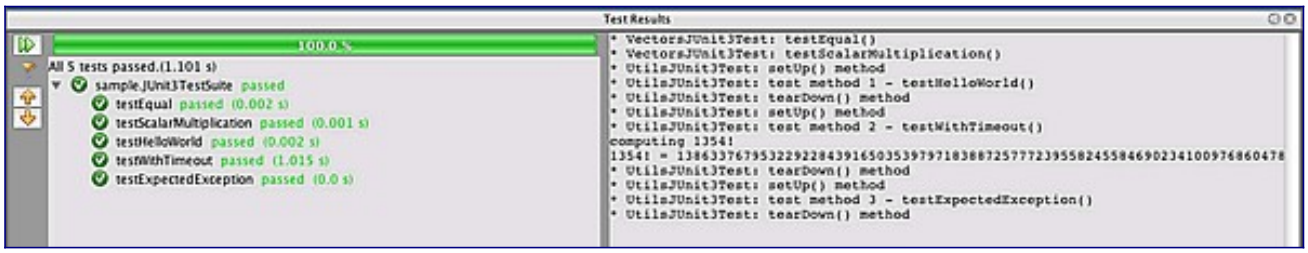
```
@RunWith(Suite.class)
@Suite.SuiteClasses(value={UtilsJUnit4Test.class, VectorsJUnit4Test.class})
public class JUnit4TestSuite {
}
```

Cuando ejecutas la suite de pruebas, la IDE ejecutará las clases de prueba en el orden en que estén listadas.

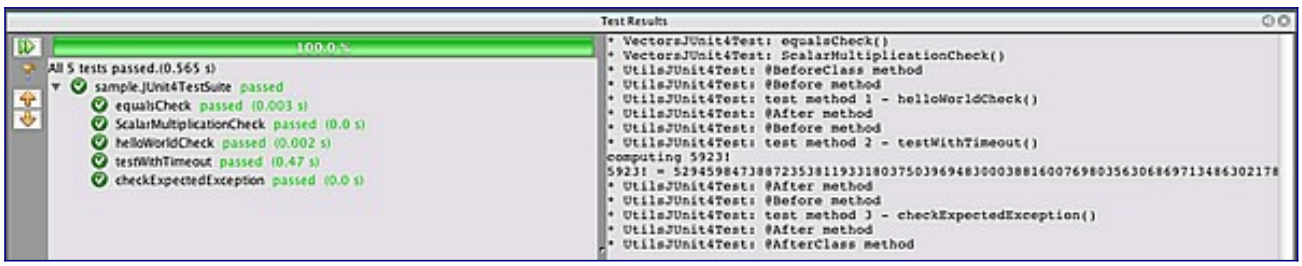
Ejecución de Suites de Pruebas

Ejecutas una suite de pruebas de la misma manera que ejecutas cualquier clase de prueba individual.
1. Expande el nodo de Paquetes de Pruebas en la ventana de Proyectos.
2. Haz clic derecho en la clase de la suite de pruebas y elige Ejecutar Archivo de Pruebas.

Cuando ejecuta el conjunto de pruebas, el IDE ejecuta las pruebas incluidas en el conjunto en el orden en que aparecen. Los resultados se muestran en la ventana Resultados de la prueba JUnit.

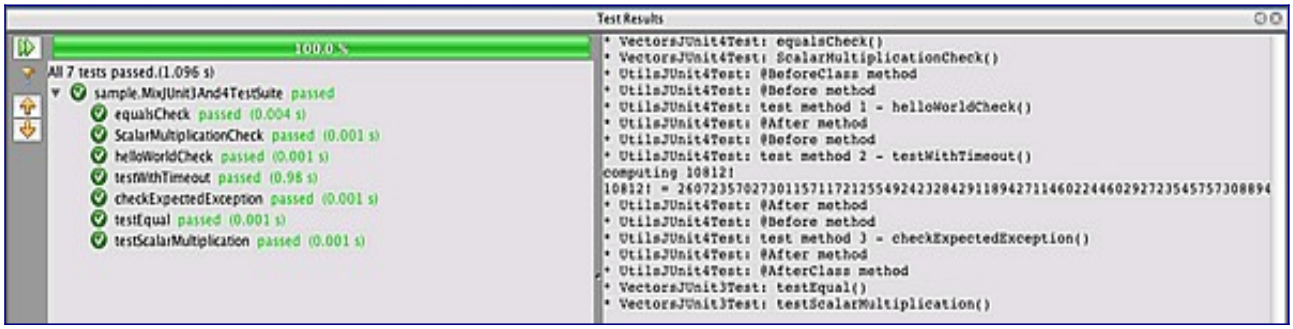


En esta imagen (haga clic en la imagen para ver una imagen más grande) puede ver los resultados de la prueba para un conjunto de pruebas JUnit 3. El conjunto de pruebas ejecutó las clases de prueba UtilsJUnit3Testy VectorsJUnit3Testcomo una sola prueba y mostró los resultados de la prueba en el panel izquierdo como los resultados de una sola prueba. El resultado en el panel derecho es el mismo que cuando ejecuta la prueba individualmente.



En esta imagen (haga clic en la imagen para ver una imagen más grande) puede ver los

resultados de la prueba para un conjunto de pruebas JUnit 4. El conjunto de pruebas ejecutó las clases de prueba UtilsJUnit4Testy VectorsJUnit4Testcomo una sola prueba y mostró los resultados de la prueba en el panel izquierdo como los resultados de una sola prueba. El resultado en el panel derecho es el mismo que cuando ejecuta la prueba individualmente.



En esta imagen (haga clic en la imagen para ver una imagen más grande) puede ver los resultados de la prueba para un conjunto de pruebas mixtas. Este conjunto de pruebas incluye el conjunto de pruebas JUnit 4 y una de las clases de prueba JUnit 3. El conjunto de pruebas ejecutó las clases de prueba UtilsJUnit3Test.javay JUnit4TestSuite.javacomo una sola prueba y mostró los resultados de la prueba en el panel izquierdo como los resultados de una sola prueba. El resultado en el panel derecho es el mismo que ejecutar la prueba individualmente.

Conclusión

Este tutorial le ha proporcionado una introducción básica a la creación de conjuntos de pruebas y pruebas unitarias JUnit en NetBeans IDE. El IDE es compatible con JUnit 3 y JUnit 4, y este documento demostró algunos de los cambios introducidos en JUnit 4 que están diseñados para simplificar la creación y ejecución de pruebas.

Como se demuestra en este tutorial, una de las principales mejoras en JUnit 4 es el soporte para anotaciones. En JUnit 4 ahora puede usar anotaciones para hacer lo siguiente:

- Identifique una prueba usando la @Testanotación en lugar de la convención de nomenclatura
- Identificar setUpy tearDownmétodos con @Beforey @Afteranotaciones
- Identificar setUpy tearDownmétodos que se aplican a toda la clase de prueba. Los métodos anotados con @BeforeClassse ejecutan solo una vez, antes de ejecutar cualquier método de prueba en la clase. Los métodos anotados con @AfterClasstambién se ejecutan solo una vez, después de que hayan finalizado todos los métodos de prueba.
- Identificar las excepciones esperadas
- Identifique las pruebas que deben omitirse utilizando la @Ignoreanotación
- Especificar un parámetro de tiempo de espera para una prueba

Para obtener más información sobre el uso de JUnit y otros cambios introducidos en JUnit 4, consulte los siguientes recursos:

- junit.org

Probar el código a menudo ayuda a garantizar que los pequeños cambios realizados en el código no rompan la aplicación. Las herramientas de prueba automatizadas como JUnit agilizan el proceso de prueba y las pruebas frecuentes pueden ayudar a detectar errores de codificación temprano.