



Calidad, Pruebas y Mantenimiento del Software

PL1



Título: Manual de EvoSuite

Integrantes

Ana Cortés Cercadillo (a.cortesc@edu.uah.es)

Carlos Javier Hellín Asensio (carlos.hellin@edu.uah.es)

Daniel Ferreiro Rodríguez (daniel.ferreiror@edu.uah.es)

Grado de Ingeniería Informática

Curso: **2021-2022**

Contenido

EvoSuite.....	3
Programa tutorial.....	3
Stack.java.....	4
StackTest.java	4
Programa Calculadora	11
Calculadora.java	11
Principal.java.....	12

EvoSuite

EvoSuite es una herramienta que genera automáticamente pruebas unitarias para Java. Utiliza un algoritmo evolutivo para generar pruebas de JUnit pudiéndolas ejecutar desde la línea de comandos. Tiene complementos para integrarlo en Maven , IntelliJ y Eclipse. EvoSuite aplica un enfoque híbrido novedoso que genera y optimiza suites de prueba completas para satisfacer un criterio de cobertura.

Principales características:

- Generación de pruebas JUnit 4 para las clases seleccionadas.
- Optimización de diferentes criterios de cobertura, como líneas, ramales, salidas y pruebas de mutación.
- Se minimizan las pruebas: solo se conservan las que contribuyen a lograr la cobertura.
- Generación de JUnit afirma capturar el comportamiento actual de las clases probadas.

Prerrequisitos

- 1- Tener instalado Java 8 JDK.

```
$ javac -version  
javac 1.8.0_312
```

- 2- Tener instalado Apache Maven (versión 3.1 o posterior).

```
$ mvn -version  
Apache Maven 3.3.9
```

Programa tutorial

El programa de ejemplo, escrito en Java, es la implementación de la estructura de datos **Stack** (pila) con sus tres métodos clásicos: **push()**, **pop()** y **isEmpty()**.

Lo primero que hacemos es descargar el proyecto:

```
$ wget http://evosuite.org/files/tutorial/Tutorial_Stack.zip
```

Descomprimir el proyecto:

```
$ unzip Tutorial_Stack.zip
```

Cambiamos de directorio:

```
$ cd Tutorial_Stack
```

El proyecto maven tiene la siguiente composición:

pom.xml: archivo de compilación maven.

src: este directorio tiene la siguiente estructura:

```
cubanito@ferrari1-VB:~/Calidad/PL1/Tutorial_Stack$ tree src
src
├── main
│   └── java
│       └── tutorial
│           └── Stack.java
└── test
    └── java
        └── tutorial
            └── StackTest.java

6 directories, 2 files
```

Stack.java

```
package tutorial;

import java.util.EmptyStackException;

public class Stack<T> {
    private int capacity = 10;
    private int pointer = 0;
    private T[] objects = (T[]) new Object[capacity];

    public void push(T o) {
        if(pointer >= capacity)
            throw new RuntimeException("Stack exceeded capacity!");
        objects[pointer++] = o;
    }
    public T pop() {
        if(pointer <= 0)
            throw new EmptyStackException();
        return objects[--pointer];
    }
    public boolean isEmpty() {
        return pointer <= 0;
    }
}
```

StackTest.java

```
package tutorial;

import org.junit.Test;
import org.junit.Assert;

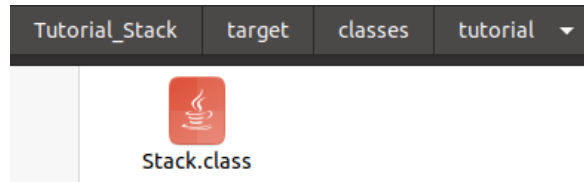
public class StackTest {

    @Test
    public void test() {
        Stack<Object> stack = new Stack<Object>();
        stack.push(new Object());
        Assert.assertFalse(stack.isEmpty());
    }
}
```

Dentro de la carpeta del proyecto, para comenzar, se invoca a **maven** para compilar el proyecto:

```
$ mvn compile
```

Maven producirá algunos resultados para informar sobre lo que está haciendo y, al final, debería ver un mensaje de éxito, en caso contrario, dará los mensajes de errores pertinentes. Como resultado de la compilación, hay un archivo en **target/classes/tutorial/Stack.class** que contiene el bytecode de la clase **Stack**.



Obtención de EvoSuite

Descargar **evosuite-1.0.6.jar** y **evosuite-standalone-runtime-1.0.6.jar** para generar las pruebas:

```
$ wget
https://github.com/EvoSuite/evosuite/releases/download/v1.0.6/evosuite-1.0.6.jar
```

```
$ wget
https://github.com/EvoSuite/evosuite/releases/download/v1.0.6/evosuite-standalone-runtime-1.0.6.jar
```

Ejecutar el ejecutable con el comando:

```
$ java -jar evosuite-1.0.6.jar
```

Para una mayor facilidad, se crea una variable de entorno para apuntar a EvoSuite con el siguiente comando:

```
$ export EVOSUITE="java -jar $(pwd)/evosuite-1.0.6.jar"
```

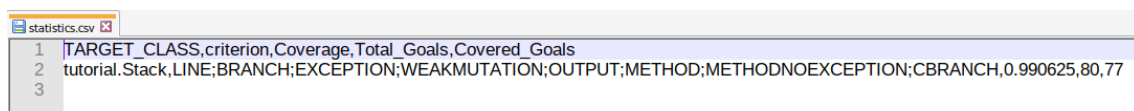
Se ejecuta EvoSuite de la siguiente forma:

```
$ $EVOSUITE -class tutorial.Stack -projectCP target/classes
```

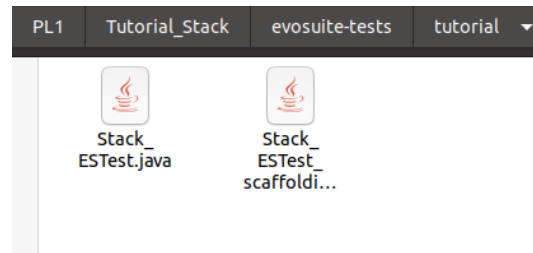
Con esta última ejecución se creó dos directorios nuevos:

- **evosuite-report**
- **evosuite-test**

El directorio **evosuite-report** tiene un archivo llamado **statistics.csv** con información de la compilación:



Si todo funcionó correctamente, entonces EvoSuite ha creado cuatro archivos en la carpeta `evo-tests`:



Nota: El archivo *scaffolding* tiene métodos con las etiquetas **@Before** y **@After**. Estas son anotaciones JUnit que aseguran que estos métodos se ejecuten antes o después de la ejecución de cada prueba individual. Las pruebas están en el archivo *Stack_ESTest.java*.

Stack_ESTest.java

```
package tutorial;

import org.junit.Test;
import static org.junit.Assert.*;
import static org.evosuite.runtime.EvoAssertions.*;
import java.util.EmptyStackException;
import org.evosuite.runtime.EvoRunner;
import org.evosuite.runtime.EvoRunnerParameters;
import org.junit.runner.RunWith;
import tutorial.Stack;

@RunWith(EvoRunner.class) @EvoRunnerParameters(mockJVMNonDeterminism = true, useVFS = true,
useVNET = true, resetStaticState = true, separateClassLoader = true, useJEE = true)
public class Stack_ESTest extends Stack_ESTest_scaffolding {

    @Test(timeout = 4000)
    public void test0() throws Throwable {
        Stack<Object> stack0 = new Stack<Object>();
        assertTrue(stack0.isEmpty());

        Object object0 = new Object();
        stack0.push(object0);
        stack0.push("");
        stack0.pop();
        assertFalse(stack0.isEmpty());
    }

    @Test(timeout = 4000)
    public void test1() throws Throwable {
        Stack<Object> stack0 = new Stack<Object>();
        stack0.push((Object) null);
        stack0.pop();
        assertTrue(stack0.isEmpty());
    }

    @Test(timeout = 4000)
    public void test2() throws Throwable {
        Stack<Object> stack0 = new Stack<Object>();
        assertTrue(stack0.isEmpty());

        Integer integer0 = new Integer((-38));
        stack0.push(integer0);
        boolean boolean0 = stack0.isEmpty();
        assertFalse(boolean0);
    }
}
```

```

@Test(timeout = 4000)
public void test3() throws Throwable {
    Stack<String> stack0 = new Stack<String>();
    // Undeclared exception!
    try {
        stack0.pop();
        fail("Expecting exception: EmptyStackException");

    } catch (EmptyStackException e) {
        //
        // no message in exception (getMessage() returned null)
        //
        verifyException("tutorial.Stack", e);
    }
}

@Test(timeout = 4000)
public void test4() throws Throwable {
    Stack<String> stack0 = new Stack<String>();
    stack0.push("<");
    assertFalse(stack0.isEmpty());

    stack0.pop();
    boolean boolean0 = stack0.isEmpty();
    assertTrue(boolean0);
}

@Test(timeout = 4000)
public void test5() throws Throwable {
    Stack<Object> stack0 = new Stack<Object>();
    Integer integer0 = new Integer((-38));
    stack0.push(integer0);
    stack0.push("Ad");
    stack0.push(integer0);
    stack0.push("Ad");
    stack0.push(integer0);
    stack0.push("Ad");
    stack0.push(integer0);
    stack0.push(integer0);
    stack0.push(integer0);
    stack0.push(integer0);
    // Undeclared exception!
    try {
        stack0.push("Ad");
        fail("Expecting exception: RuntimeException");

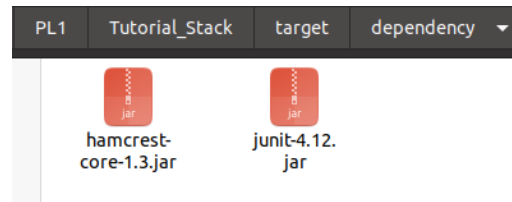
    } catch (RuntimeException e) {
        //
        // Stack exceeded capacity!
        //
        verifyException("tutorial.Stack", e);
    }
}
}

```

Ejecución de pruebas en EvoSuite

Para obtener las dependencias de *junit* y *hamcrest*, podemos usar maven con el siguiente comando:

```
$ mvn dependency:copy-dependencies
```



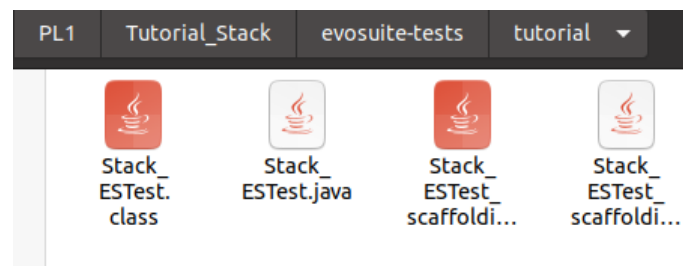
Ahora necesitamos decirle al compilador de Java dónde encontrar todas estas cosas, para lo cual configuramos la variable de entorno **CLASSPATH**:

```
$ export CLASSPATH=target/classes:evosuite-standalone-runtime-1.0.6.jar:evosuite-tests:target/dependency/junit-4.12.jar:target/dependency/hamcrest-core-1.3.jar
```

Se integra EvoSuite en el proyecto maven de modo que este último se encarga de compilar las pruebas:

```
$ javac evosuite-tests/tutorial/*.java
```

Compruebe que hay dos archivos **.class** en **evosuite-tests/tutorial**. Si no están allí, se debe verificar los mensajes de error que dio el compilador de Java.



Ejecutar las pruebas en la línea de comandos:

```
$ java org.junit.runner.JUnitCore tutorial.Stack_ESTest
JUnit version 4.12
.....
Time: 0.386
OK (6 tests)
```

Se generó y se ejecutó el primer conjunto de pruebas de EvoSuite.

Trabajar con pruebas existentes

El archivo de pruebas que se encuentra en la ruta `src/test/java/tutorial/StackTest.java`

contiene una única prueba. Con la propiedad usando `-Djunit=tutorial.StackTest` nos proporcionaría pruebas que aún no están cubiertas por `tutorial.StackTest`.

EvoSuite necesita encontrar y ejecutar esa prueba para ello se usa maven:

```
$ mvn test
```

```
-----  
T E S T S  
-----
```

```
Running tutorial.StackTest
```

```
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.076 sec
```

```
Results :
```

```
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

Para EvoSuite, la parte interesante es que maven colocó el código de bytes de esta prueba en el directorio ***target/test-classes/tutorial/StackTest.class***. Para saber qué tan bueno es este conjunto de pruebas, podemos pedirle a EvoSuite que mida la cobertura. EvoSuite admite el comando `-measureCoverage`, y necesitamos especificar la clase bajo prueba (***-class tutorial.Stack***), las pruebas que nos interesan (***-Djunit=tutorial.StackTest***), el classpath que contiene la clase bajo prueba y las pruebas (***-projectCP target/classes:target/test-classes***):

```
$ $EVOSUITE -measureCoverage -class tutorial.Stack -  
Djunit=tutorial.StackTest -criterion branch -projectCP  
target/classes:target/test-classes
```

```
cubanito@ferrari-VB:~/Calidad/PL1/Tutorial_Stack$ $EVOSUITE -measureCoverage -class tutorial.Stack  
-Djunit=tutorial.StackTest -criterion branch -projectCP target/classes:target/test-classes  
* EvoSuite 1.0.6  
* Starting client  
* Connecting to master process on port 3044  
  - target/classes  
  - target/test-classes  
* Finished analyzing classpath  
* Analyzing entry: tutorial.StackTest  
* Found 1 test class(es)  
* Executing test(s)  
  Executing StackTest  
* Number of test cases to execute: 1  
* Started: ClassName: tutorial.StackTest, MethodName: test  
* Finished: ClassName: tutorial.StackTest, MethodName: test  
* Number of test cases executed: 1  
* Executed 1 unit test(s)  
* Target class tutorial.Stack  
* Coverage of criterion BRANCH: 43%  
* Number of covered goals: 3 / 7  
* Total number of covered goals: 3 / 7  
* Total coverage: 43%  
* Computation finished  
* Writing statistics
```

Si se quiere tener pruebas que cubran los 4 objetivos de cobertura restantes, hay que invocar a EvoSuite de la siguiente manera:

```
$ $EVOSUITE -class tutorial.Stack -Djunit=tutorial.StackTest -projectCP
target/classes:target/test-classes -criterion branch
```

```
cubanito@ferrari-VB:~/Calidad/PL1/Tutorial_Stack$ $EVOSUITE -class tutorial.Stack -Djunit=tutorial.StackTest -
projectCP target/classes:target/test-classes -criterion branch
* EvoSuite 1.0.6
* Going to generate test cases for class: tutorial.Stack
* Starting client
* Connecting to master process on port 8285
* Analyzing classpath:
  - target/classes
  - target/test-classes
* Finished analyzing classpath
* Generating tests for class tutorial.Stack
* Test criterion:
  - Branch Coverage
* Setting up search algorithm for whole suite generation
* Total number of test goals: 7
* Using seed 1649000264029
* Starting evolution
[Progress:>                                     3%] [Cov:=====100%]
* Search finished after 3s and 20 generations, 7391 statements, best individual has fitness: 0.0
* Minimizing test suite
* Determining coverage of existing tests
  Executing StackTest
* Number of test cases to execute: 1
* Started: ClassName: tutorial.StackTest, MethodName: test
* Finished: ClassName: tutorial.StackTest, MethodName: test
* Number of test cases executed: 1
* Executed 1 unit test(s)
* Removing 3 goals already covered by JUnit (total: 7)
* Going to analyze the coverage criteria
* Coverage analysis for criterion BRANCH
* Coverage of criterion BRANCH: 86%
* Total number of goals: 7
* Number of covered goals: 6
* Generated 4 tests with total length 20
* Resulting test suite's coverage: 100%
* Generating assertions
* Resulting test suite's mutation score: 53%
* Compiling and checking tests
* Writing JUnit test case 'Stack_ESTest' to evosuite-tests
* Done!

* Computation finished
```

Programa Calculadora

Se crea un pequeño programa en Java que simulará una calculadora, permitiendo la entrada por teclado de dos números y un operador (+-*/*), permitiendo cuatro operaciones básicas (sumar, restar, multiplicar o dividir).

El proyecto de maven tiene 2 clases: **Calculadora.java** y **Principal.java**

Calculadora.java

La clase tiene un método principal llamado operación que recibe dos números y un operador devolviendo un resultado. Casos posibles:

- Si es el operador está permitido (+-*/*) realiza la operación correspondiente y se retorna el resultado.
- Si el operador no está permitido el método retorna un mensaje indicándolo.

Nota: Si el divisor (num2) y el operador es '/', se retorna un mensaje indicando que la división por 0 no está permitida.

```
public class Calculadora
{
    public Calculadora() { }

    public String operacion(double num1, double num2, char operador)
    {
        String resultado = "";
        switch(operador){
            case '+':
                resultado = num1 + " + " + num2 + " = " + (num1+num2);
                break;
            case '-':
                resultado = num1 + " - " + num2 + " = " + (num1-num2);
                break;
            case '*':
                resultado = num1 + " * " + num2 + " = " + (num1*num2);
                break;
            case '/':
                resultado = (num2 != 0)? (num1 + " / " + num2 + " = " + (num1/num2)): "La division por 0 no esta permitida";
                break;
            default:
                resultado = "El operador " + operador + " no esta permitido";
                break;
        }
        return resultado;
    }
}
```

Principal.java

Se instancia el objeto de la clase Calculadora, se introduce en el método **operacion** dos números y el operador introducidos por teclado. Este método devuelve una respuesta, a continuación se pregunta si se quiere seguir y el programa termina solo si se escribe la letra 's'.

```
import java.util.Scanner;

public class Principal
{
    public static void main(String [] args)
    {
        Calculadora calculadora = new Calculadora();
        Scanner scanner = new Scanner(System.in);
        boolean salir;

        do {

            try {
                System.out.print("Primer numero: "); double num1 = Double.parseDouble(scanner.next());
                System.out.print("Operador (+-* /): "); char operador = scanner.next().charAt(0);
                System.out.print("Segundo numero: "); double num2 = Double.parseDouble(scanner.next());

                System.out.println(calculadora.operacion(num1, num2, operador));
            } catch (NumberFormatException e) {
                System.out.println("No es un número.");
            }

            System.out.print("Presiona 's' si quieres terminar o cualquier otra tecla para continuar: ");
            String respuesta = scanner.next();
            salir = (respuesta.equals("s"))? true : false;

        }while (!salir);
    }
}
```

Ejemplo de ejecución

```
Primer numero: 3
Operador (+-* /): *
Segundo numero: 2
3.0 * 2.0 = 6.0
Presiona 's' si quieres terminar o cualquier otra tecla para continuar: m
Primer numero: 4.7
Operador (+-* /): /
Segundo numero: 0
La division por 0 no esta permitida
Presiona 's' si quieres terminar o cualquier otra tecla para continuar: n
Primer numero: 3
Operador (+-* /): ,
Segundo numero: 1
El operador , no esta permitido
Presiona 's' si quieres terminar o cualquier otra tecla para continuar: s

Process finished with exit code 0
```

En el proyecto, el archivo **pom.xml** se le añaden las siguientes líneas remarcadas en rojo para añadir las dependencias a JUnit para ejecutar las pruebas.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>Calculador</artifactId>
  <version>1.0-SNAPSHOT</version>

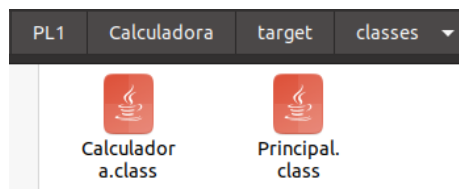
  <name>Calculador</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Dentro de la carpeta del proyecto, para comenzar, se invoca a maven para compilar el proyecto:

```
$ mvn compile
```

Maven producirá algunos resultados para informar sobre lo que está haciendo y, al final, debería ver un mensaje de éxito, en caso contrario, dará los mensajes de errores pertinentes.



Obtención de EvoSuite

Descargar **evosuite-1.0.6.jar** y **evosuite-standalone-runtime-1.0.6.jar** para generar las pruebas

```
wget
```

<https://github.com/EvoSuite/evosuite/releases/download/v1.0.6/evosuite-1.0.6.jar>

```
wget
```

<https://github.com/EvoSuite/evosuite/releases/download/v1.0.6/evosuite-standalone-runtime-1.0.6.jar>

Ejecutar el ejecutable con el comando:

```
$ java -jar evosuite-1.0.6.jar
```

Para una mayor facilidad, se crea una variable de entorno para apuntar a EvoSuite con el siguiente comando:

```
$ export EVOSUITE="java -jar $(pwd)/evosuite-1.0.6.jar"
```

Se puede ejecutar de forma individual:

```
$ $EVOSUITE -class Calculadora -projectCP target/classes
$ $EVOSUITE -class Principal -projectCP target/classes
```

O

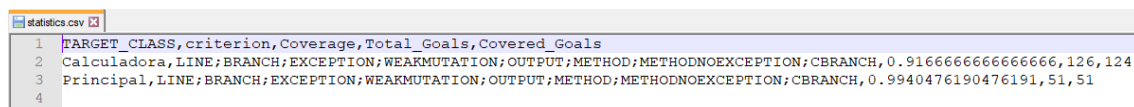
Para invocar EvoSuite en todas las clases:

```
$ $EVOSUITE -target target/classes
```

Con esta última ejecución se creó dos directorios nuevos:

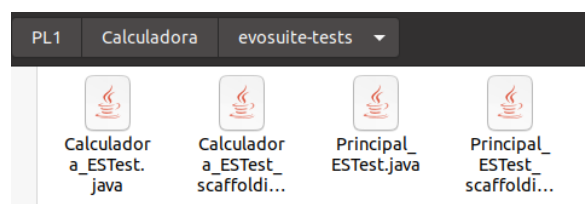
- **evosuite-report**
- **evosuite-test**

El directorio **evosuite-report** tiene un archivo llamado **statistics.csv** con información de la compilación:



```
statistics.csv
1 TARGET_CLASS,criterion,Coverage,Total_Goals,Covered_Goals
2 Calculadora,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.9166666666666666,126,124
3 Principal,LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH,0.9940476190476191,51,51
4
```

Si todo funcionó correctamente, entonces EvoSuite ha creado cuatro archivos en la carpeta **evosuite-tests**:



Nota: Los archivos **scaffolding** tienen métodos con las etiquetas **@Before** y **@After**. Estas son anotaciones JUnit que aseguran que estos métodos se ejecuten antes o después de la ejecución de cada prueba individual. Las pruebas están en los archivos ***_ESTest.java**.

Calculadora_ESTest.java

```
import org.junit.Test;
import static org.junit.Assert.*;
import org.evosuite.runtime.EvoRunner;
import org.evosuite.runtime.EvoRunnerParameters;
import org.junit.runner.RunWith;

@RunWith(EvoRunner.class) @EvoRunnerParameters(mockJVMNonDeterminism = true, useVFS = true,
useVNET = true, resetStaticState = true, separateClassLoader = true, useJEE = true)
public class Calculadora_ESTest extends Calculadora_ESTest_scaffolding {

    @Test(timeout = 4000)
    public void test0() throws Throwable {
        Calculadora calculadora0 = new Calculadora();
        String string0 = calculadora0.operacion(1.0, (-1477.6), '/');
        assertEquals("1.0 / -1477.6 = -6.76773145641581E-4", string0);
    }

    @Test(timeout = 4000)
    public void test1() throws Throwable {
        Calculadora calculadora0 = new Calculadora();
        String string0 = calculadora0.operacion(1781.41883, 1765.912717325, '*');
        assertEquals("1781.41883 * 1765.912717325 = 3145830.166779222", string0);
    }

    @Test(timeout = 4000)
    public void test2() throws Throwable {
        Calculadora calculadora0 = new Calculadora();
        String string0 = calculadora0.operacion(0.0, 1174.07, '-');
        assertEquals("0.0 - 1174.07 = -1174.07", string0);
    }

    @Test(timeout = 4000)
    public void test3() throws Throwable {
        Calculadora calculadora0 = new Calculadora();
        String string0 = calculadora0.operacion((-950.90117), 0.0, '+');
        assertEquals("-950.90117 + 0.0 = -950.90117", string0);
    }

    @Test(timeout = 4000)
    public void test4() throws Throwable {
        Calculadora calculadora0 = new Calculadora();
        String string0 = calculadora0.operacion((-3070.559), 0.0, '/');
        assertEquals("La division por 0 no est\u00E1 permitida", string0);
    }

    @Test(timeout = 4000)
    public void test5() throws Throwable {
        Calculadora calculadora0 = new Calculadora();
        String string0 = calculadora0.operacion(1633.33666304, 1056.5385418, '&');
        assertEquals("El operador & no esta permitido", string0);
    }

    @Test(timeout = 4000)
    public void test6() throws Throwable {
        Calculadora calculadora0 = new Calculadora();
        String string0 = calculadora0.operacion('/', '/', '/');
        assertEquals("47.0 / 47.0 = 1.0", string0);
    }

    @Test(timeout = 4000)
    public void test7() throws Throwable {
        Calculadora calculadora0 = new Calculadora();
        String string0 = calculadora0.operacion('.', '.', '.');
        assertEquals("El operador . no esta permitido", string0);
    }

    @Test(timeout = 4000)
    public void test8() throws Throwable {
        Calculadora calculadora0 = new Calculadora();
        String string0 = calculadora0.operacion(12.15483475352594, 12.15483475352594, ',');
        assertEquals("El operador , no esta permitido", string0);
    }

    @Test(timeout = 4000)
    public void test9() throws Throwable {
        Calculadora calculadora0 = new Calculadora();
        String string0 = calculadora0.operacion(1.0, 1.0, '+');
        assertEquals("1.0 + 1.0 = 2.0", string0);
    }
}
```

Principal_ESTest.java

```
import org.junit.Test;
import static org.junit.Assert.*;
import static org.evosuite.runtime.EvoAssertions.*;
import java.util.NoSuchElementException;
import org.evosuite.runtime.EvoRunner;
import org.evosuite.runtime.EvoRunnerParameters;
import org.evosuite.runtime.util.SystemInUtil;
import org.junit.runner.RunWith;

@RunWith(EvoRunner.class) @EvoRunnerParameters(mockJVMNonDeterminism = true, useVFS = true,
useVNET = true, resetStaticState = true, separateClassLoader = true, useJEE = true)
public class Principal_ESTest extends Principal_ESTest_scaffolding {

    @Test(timeout = 4000)
    public void test0() throws Throwable {
        String[] stringArray0 = new String[4];
        SystemInUtil.addInputLine("4");
        SystemInUtil.addInputLine("3");
        SystemInUtil.addInputLine("3");
        // Undeclared exception!
        try {
            Principal.main(stringArray0);
            fail("Expecting exception: NoSuchElementException");

        } catch (NoSuchElementException e) {
            //
            // no message in exception (getMessage() returned null)
            //
            verifyException("java.util.Scanner", e);
        }
    }

    @Test(timeout = 4000)
    public void test1() throws Throwable {
        String[] stringArray0 = new String[1];
        SystemInUtil.addInputLine("4");
        SystemInUtil.addInputLine("s");
        SystemInUtil.addInputLine("4");
        SystemInUtil.addInputLine("s");
        Principal.main(stringArray0);
        assertEquals(1, stringArray0.length);
    }

    @Test(timeout = 4000)
    public void test2() throws Throwable {
        String[] stringArray0 = new String[13];
        SystemInUtil.addInputLine("t");
        SystemInUtil.addInputLine("t");
        // Undeclared exception!
        try {
            Principal.main(stringArray0);
            fail("Expecting exception: NoSuchElementException");

        } catch (NoSuchElementException e) {
            //
            // no message in exception (getMessage() returned null)
            //
            verifyException("java.util.Scanner", e);
        }
    }

    @Test(timeout = 4000)
    public void test3() throws Throwable {
        Principal principal0 = new Principal();
    }
}
```

Ejecución de pruebas de EvoSuite

Para obtener las dependencias de *junit* y *hamcrest*, podemos usar maven con el siguiente comando:

```
$ mvn dependency:copy-dependencies
```

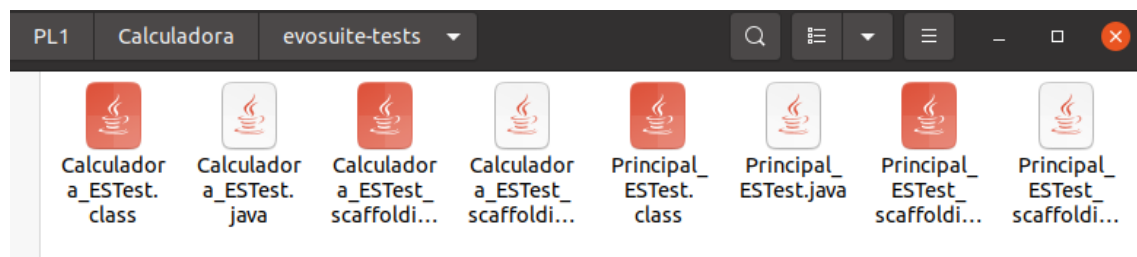

Ahora necesitamos decirle al compilador de Java dónde encontrar todas estas cosas, para lo cual configuramos la variable de entorno CLASSPATH:

```
$ export CLASSPATH=target/classes:evosuite-standalone-runtime-1.0.6.jar:evosuite-tests:target/dependency/junit-4.12.jar:target/dependency/hamcrest-core-1.3.jar
```

Se integra EvoSuite en el proyecto Maven de modo que este último se encarga de compilar las pruebas:

```
$ javac evosuite-tests/*.java
```

Compruebe que hay cuatro archivos **.class** en **evosuite-tests**. Si no están allí, se debe verificar los mensajes de error que dio el compilador de Java.



Ejecutar las pruebas en la línea de comandos:

```
$ java org.junit.runner.JUnitCore Calculadora_ESTest
JUnit version 4.12
.....
Time: 0.451
OK (10 tests)
```

```
$ java org.junit.runner.JUnitCore Principal_ESTest
JUnit version 4.12
.....
Time: 0.39
OK (3 tests)
```