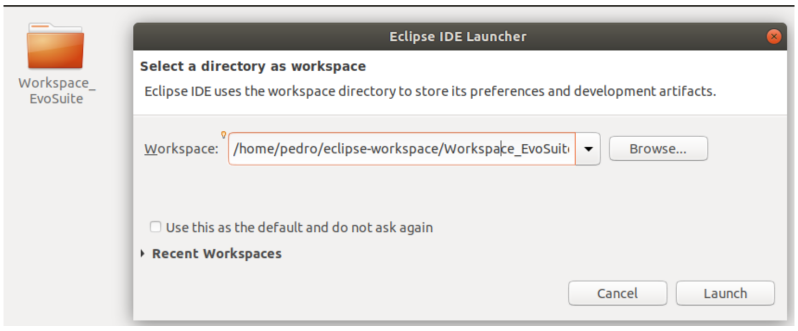
SURVEY ON INTERACTIVE TEST CASE GENERATION: TUTORIAL

Thanks for participating in our experiment. The following steps will guide you through the interactive execution of the test case generation tool EvoSuite. For this first part of the experiment, we will use the implementation of a simple class called `Stack.java`, which will be our sample class under test.

Preparing EvoSuite and the class under test

1) Open Eclipse IDE. When asked about the workspace, select the existing workspace and click Launch.



Eclipse IDE Launcher

Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: /home/pedro/eclipse-workspace/Workspace_EvoSuit

☐ Use this as the default and do not ask again

Recent Workspaces

2) Once Eclipse IDE is ready for use, click Navigate / Open resource… in the menu. In the popup window, type `Stack.java`. You should see two entries: select the file in “evosuite-master” and click Open:

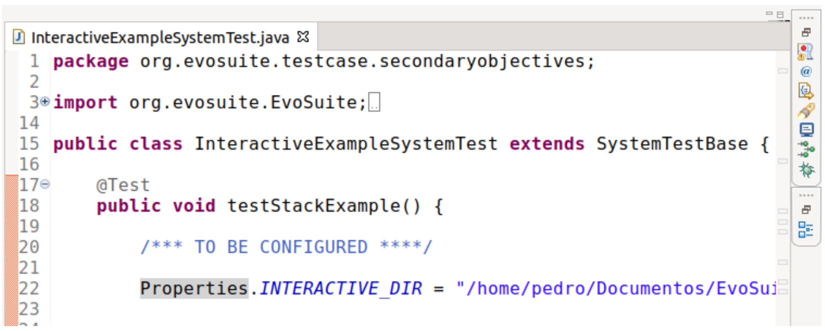
```java
package tutorial;

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;
    }
}
```

As it can be observed, it is a simple Java class representing a stack data structure, that counts with 3 private attributes (capacity, pointer and objects) and 3 methods (push, pop and isEmpty).

3) In the menu, click Navigate / Open resource… In the popup window, type `InteractiveExampleSystemTest.java`. You should see two entries: select the file in “evosuite-master” and click Open:

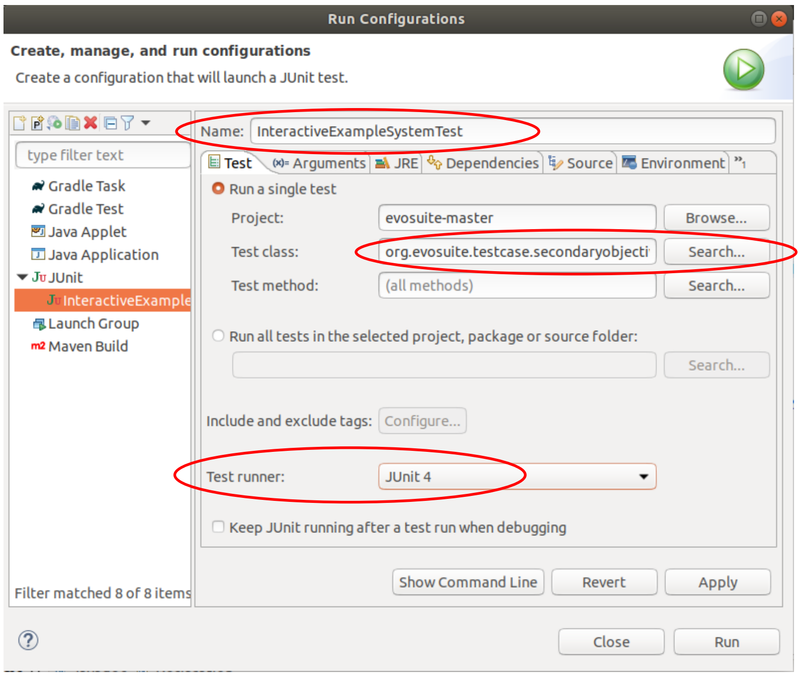
Note: Please, do not modify the files unless instructed to do so.

Result:



```
package org.evosuite.testcase.secondaryobjectives;

import org.evosuite.EvoSuite;

public class InteractiveExampleSystemTest extends SystemTestBase {

    @Test
    public void testStackExample() {

        /*** TO BE CONFIGURED ****/

        Properties.INTERACTIVE_DIR = "/home/pedro/Documentos/EvoSui
```

4) Finally, click “Run / Run configurations…”:
   - Make sure that the “Test class” is *InteractiveExampleSystemTest*. Otherwise, click “Search…” and select this class.
   - Change the “Test runner” to JUnit 4:



Run Configurations

Create, manage, and run configurations

Create a configuration that will launch a JUnit test.

Name: InteractiveExampleSystemTest

Run a single test

Project: evosuite-master

Test class: org.evosuite.testcase.secondaryobjecti

Test method: (all methods)

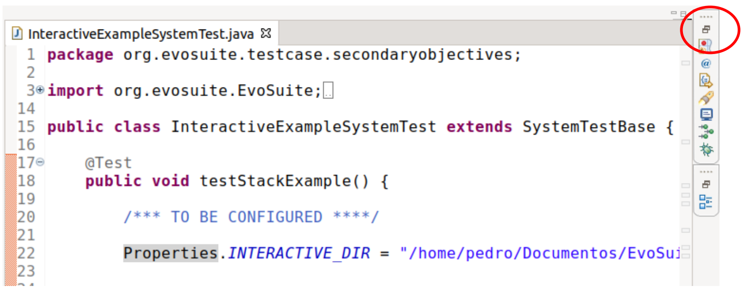
Test runner: JUnit 4

Click “Apply” and then click “Close”.

Execution

Please, follow the next steps that will guide you through the interactive execution of EvoSuite for the generation of a test suite for the class `Stack.java`

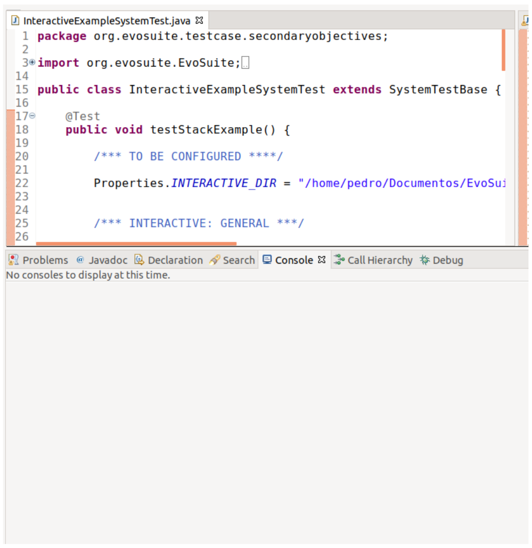
1) **Console**: Before the execution, make sure that the Console window is not minimized. Otherwise, restore it to be able to observe the messages of the execution (*click on the button rounded with a red circle*).



```
package org.evosuite.testcase.secondaryobjectives;

import org.evosuite.EvoSuite;

public class InteractiveExampleSystemTest extends SystemTestBase {

    @Test
    public void testStackExample() {

        /*** TO BE CONFIGURED ****/

        Properties.INTERACTIVE_DIR = "/home/pedro/Documentos/EvoSui
```

Note: If you cannot find the console, click Window / Show View / Console in the menu.

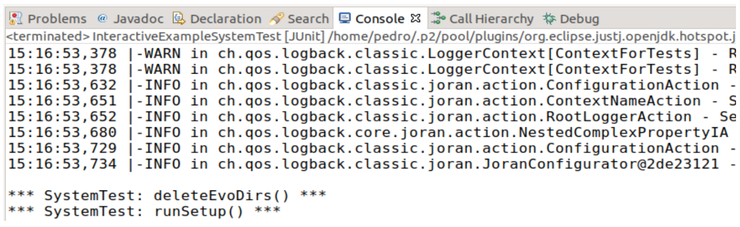
For instance, you can place the Console at the bottom taking half of the screen.



```
package org.evosuite.testcase.secondaryobjectives;

import org.evosuite.EvoSuite;

public class InteractiveExampleSystemTest extends SystemTestBase {

    @Test
    public void testStackExample() {

        /*** TO BE CONFIGURED ****/

        Properties.INTERACTIVE_DIR = "/home/pedro/Documentos/EvoSui

        /*** INTERACTIVE: GENERAL ***/
```

```
No consoles to display at this time.
```

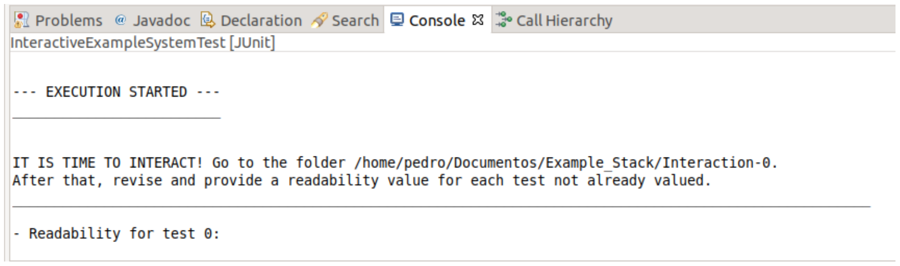
2) **Starting the execution**: It is time to generate test cases interactively with EvoSuite! Click somewhere in the file `InteractiveExampleSystemTest.java` and then click "Run / Run" in the menu. After some seconds, you should see something like this:



```
<terminated> InteractiveExampleSystemTest [JUnit] /home/pedro/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.j
15:16:53,378 |-WARN in ch.qos.logback.classic.LoggerContext[ContextForTests] - R
15:16:53,378 |-WARN in ch.qos.logback.classic.LoggerContext[ContextForTests] - R
15:16:53,632 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction -
15:16:53,651 |-INFO in ch.qos.logback.classic.joran.action.ContextNameAction - S
15:16:53,652 |-INFO in ch.qos.logback.classic.joran.action.RootLoggerAction - Se
15:16:53,680 |-INFO in ch.qos.logback.core.joran.action.NestedComplexPropertyIA
15:16:53,729 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction -
15:16:53,734 |-INFO in ch.qos.logback.classic.joran.JoranConfigurator@2de23121 -

*** SystemTest: deleteEvoDirs() ***
*** SystemTest: runSetup() ***
```

Note: Please, note that, from now on, your own execution can differ from what is shown in this tutorial (number of interactions, test cases selected, goals…). The exact details do not matter while you understand the process.

3) **First interaction:** The execution has just started. At some point during the search performed by EvoSuite, the system will request your collaboration, as shown below:



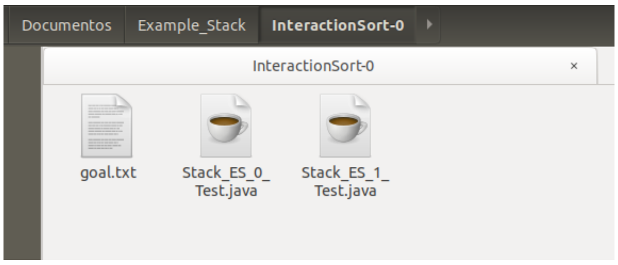
```
InteractiveExampleSystemTest [JUnit]

--- EXECUTION STARTED ---
_________________________

IT IS TIME TO INTERACT! Go to the folder /home/pedro/Documentos/Example_Stack/Interaction-0.
After that, revise and provide a readability value for each test not already valued.
_________________________________________________________________________________________

- Readability for test 0:
```

As you can observe, the first interaction (number 0) has just started and it is time for you to act. EvoSuite has selected some test cases and is waiting for you to provide your opinion on their readability.

But don't worry and take your time! The system is paused and will not resume the execution until you decide and provide your readability scores.

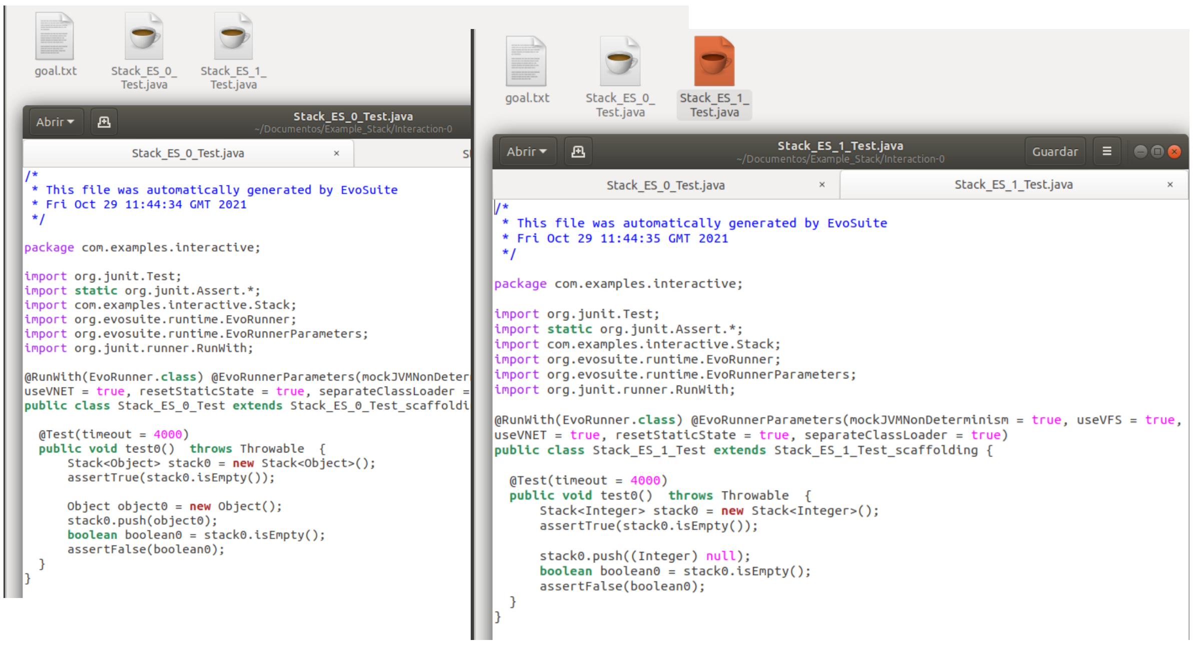
4) **Output files**: It is the moment to revise some test cases and give your opinion! Go to the directory indicated in the console. You should see:

- Some *Java* files (a maximum of 3) with the name `Stack_ES_X_Test.java`, where X is a number, starting in zero, that identifies each test case.
- A single *txt* file called `goal.txt`



The next two steps will analyze these files.

5) **Analyzing the test cases:** Each *Java* file contains a single test case that you are expected to revise. The two test cases, created with *JUnit,* look like this:

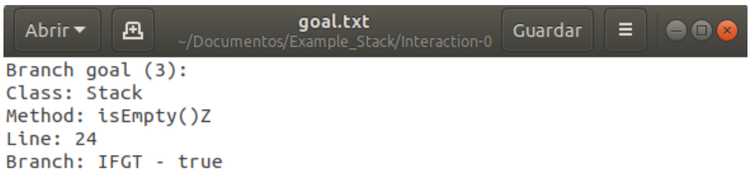


```java
/*
 * This file was automatically generated by EvoSuite
 * Fri Oct 29 11:44:34 GMT 2021
 */

package com.examples.interactive;

import org.junit.Test;
import static org.junit.Assert.*;
import com.examples.interactive.Stack;
import org.evosuite.runtime.EvoRunner;
import org.evosuite.runtime.EvoRunnerParameters;
import org.junit.runner.RunWith;

@RunWith(EvoRunner.class) @EvoRunnerParameters(mockJVMNonDeter
useVNET = true, resetStaticState = true, separateClassLoader =
public class Stack_ES_0_Test extends Stack_ES_0_Test_scaffoldi

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

      Object object0 = new Object();
      stack0.push(object0);
      boolean boolean0 = stack0.isEmpty();
      assertFalse(boolean0);
  }
}
```

```java
/*
 * This file was automatically generated by EvoSuite
 * Fri Oct 29 11:44:35 GMT 2021
 */

package com.examples.interactive;

import org.junit.Test;
import static org.junit.Assert.*;
import com.examples.interactive.Stack;
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)
public class Stack_ES_1_Test extends Stack_ES_1_Test_scaffolding {

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

      stack0.push((Integer) null);
      boolean boolean0 = stack0.isEmpty();
      assertFalse(boolean0);
  }
}
```

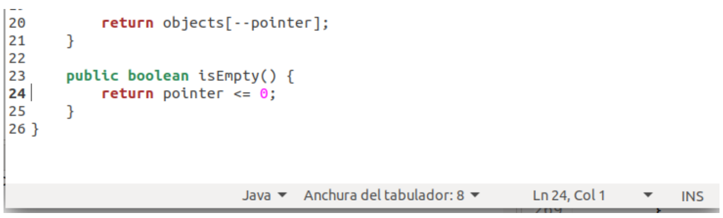
Which one, `Stack_ES_0_Test` or `Stack_ES_1_Test`, is more legible in your opinion? The system prepared the interaction to know your opinion on the readability of these test cases. This can help EvoSuite to prioritize a test case over others and even to transfer its features to the new generation of test cases.

6) **Analyzing the goal**: To help you decide, you may want to observe the goal contained in file `goal.txt`.



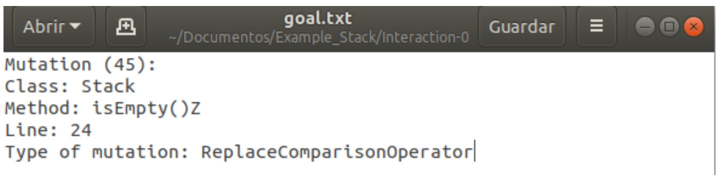
```
Branch goal (3):
Class: Stack
Method: isEmpty()Z
Line: 24
Branch: IFGT - true
```

In this particular case, EvoSuite selected these two test cases because they both cover a branch of the code (identified by the number 3), located in the method `isEmpty()` in line 24 (see code of this method below). More specifically, the branch was reached in both test cases because the condition "`pointer` greater than (GT) 0" is `true.`



```
20          return objects[--pointer];
21      }
22
23      public boolean isEmpty() {
24          return pointer <= 0;
25      }
26 }
```

Let's now see an example of a goal of type "Mutation". Mutations are just small syntactic changes purposely injected into the code.



```
Mutation (45):
Class: Stack
Method: isEmpty()Z
Line: 24
Type of mutation: ReplaceComparisonOperator
```

In this example, identified as "Mutation number 45":

- *Where in the code was inserted the mutation?* In the method `isEmpty`, line 24.
- *What kind of mutation was inserted?* The system replaced the comparison operator, for instance, by changing "`pointer <= 0`" to "`pointer > 0`".

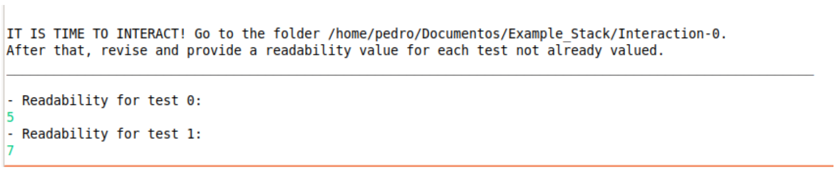
7) **Readability scores:** Have you made up your mind? It is about time to let EvoSuite know.

Go back to Eclipse and introduce your readability scores for each test case. For instance, being the range of values 0 (worst readability) to 10 (best readability), let's assign the following scores:

`Stack_ES_0_Test` = 5
`Stack_ES_1_Test` = 7

This would mean that `Stack_ES_1_Test` is more readable in our opinion than `Stack_ES_0_Test` (please, notice that this is just an example and the scores are random; each one can have a different perception of the readability of the test cases).

Result:



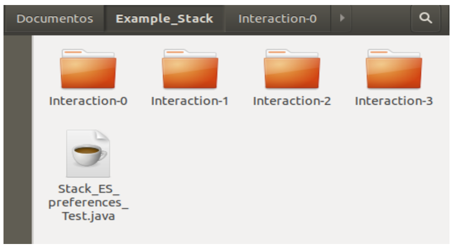
```
IT IS TIME TO INTERACT! Go to the folder /home/pedro/Documentos/Example_Stack/Interaction-0.
After that, revise and provide a readability value for each test not already valued.

- Readability for test 0:
5
- Readability for test 1:
7
```

Note 1: In case of ties, one of the tied test cases will be selected.

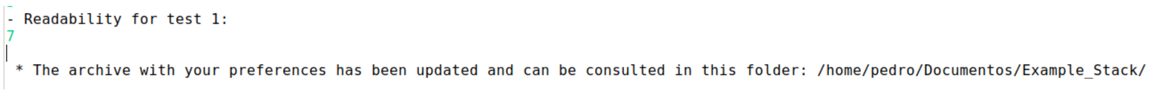
Note 2: Test cases with scores under 3 will not be considered as sufficiently readable and will not be transferred to your preference archive; this archive will be explained in step 9.

8) **Resuming the execution**: At that moment, the execution will resume and new interactions could be requested (a maximum of 6 in this example). The files corresponding to these new interactions will be placed in new folders with the name *Interaction-X*, where X represents the number of the interaction.

Note: It is recommendable to close previous output files to avoid confusion between files of different interactions.

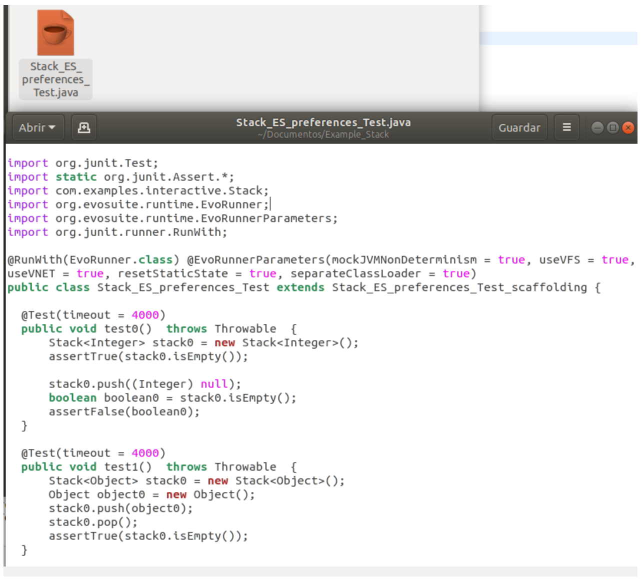


9) **Preference archive**: At this point, you might be wondering how can I know if my opinion is being taken into account? That is exactly the purpose of the *preference archive*. After some of the interaction moments, the system will inform that the preference archive has been updated:



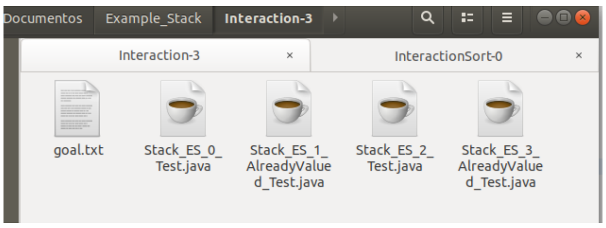
```
- Readability for test 1:
7

 * The archive with your preferences has been updated and can be consulted in this folder: /home/pedro/Documentos/Example_Stack/
```

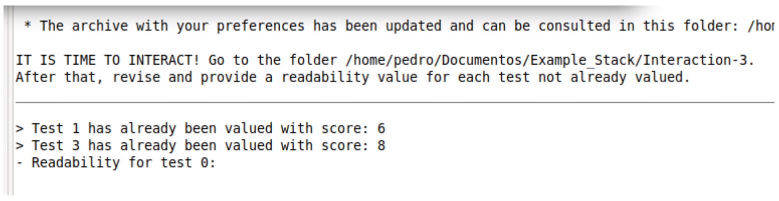
At that moment, a *Java* file with name `Stack_ES_preferences_Test.java` will appear in the output directory (see image in step 8). That file is the preference archive, which will contain the best test cases so far according to your evaluation:


Stack_ES_preferences_Test.java
Abrir
Stack_ES_preferences_Test.java
~/Documentos/Example_Stack
Guardar

```java
import org.junit.Test;
import static org.junit.Assert.*;
import com.examples.interactive.Stack;
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)
public class Stack_ES_preferences_Test extends Stack_ES_preferences_Test_scaffolding {

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

      stack0.push((Integer) null);
      boolean boolean0 = stack0.isEmpty();
      assertFalse(boolean0);
  }

  @Test(timeout = 4000)
  public void test1()  throws Throwable  {
      Stack<Object> stack0 = new Stack<Object>();
      Object object0 = new Object();
      stack0.push(object0);
      stack0.pop();
      assertTrue(stack0.isEmpty());
  }
```

10) **A new interaction (repeated tests):** Look now at what happens in a later interaction. The system informs that some of the candidate test cases for the selected covered goal were already revised in a previous interaction (and their associated readability score is also shown):



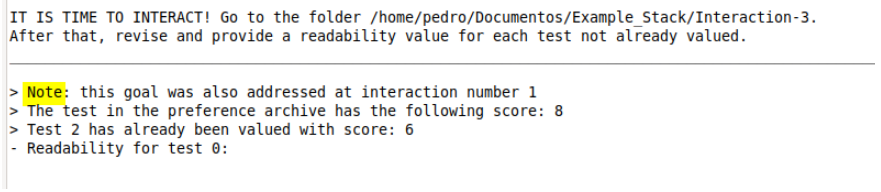


```
 * The archive with your preferences has been updated and can be consulted in this folder: /hor

IT IS TIME TO INTERACT! Go to the folder /home/pedro/Documentos/Example_Stack/Interaction-3.
After that, revise and provide a readability value for each test not already valued.

> Test 1 has already been valued with score: 6
> Test 3 has already been valued with score: 8
- Readability for test 0:
```

This is possible because the system saves memory: that means that the system will not prompt you to revise the same test case twice in case it appears again. In the output folder, the name of these tests is `Stack_ES_`**`AlreadyValued`**`_X_Test.java`, where X is a number that identifies the test case. These test cases may serve as a reference for the evaluation of the new tests, that is, `Stack_ES_0_Test.java` and `Stack_ES_2_Test.java`.

Note, however, that the already valued tests are also candidates to be associated as the most readable tests for the selected goal –if one of them has the best score once the new test cases have been valued.

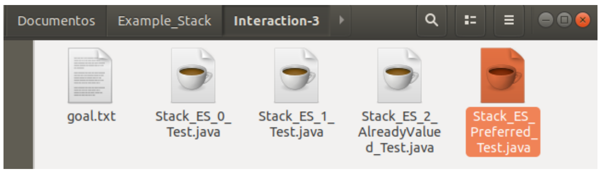
11) **A new interaction (repeated goal)**: Some other situations may arise in later interactions. In this case, the system informs that the same goal was addressed in a previous interaction:



```
IT IS TIME TO INTERACT! Go to the folder /home/pedro/Documentos/Example_Stack/Interaction-3.
After that, revise and provide a readability value for each test not already valued.

> Note: this goal was also addressed at interaction number 1
> The test in the preference archive has the following score: 8
> Test 2 has already been valued with score: 6
- Readability for test 0:
```

This means that the system requests your collaboration regarding an already addressed goal. Thanks to the note, now you can compare the new selected test cases with those already revised in the first interaction and, thus, provide a readability score for the new test cases according to your previous scores for the same goal.

The best test case in the preference archive will be shown in the output folder with the name `Stack_ES_`**`Preferred`**`_Test.java`:

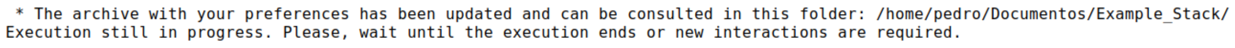


Two situations are possible after one interaction:

- If one of the new test cases (i.e., those revised in the current interaction) receives the best score, it will be inserted into the preference archive or will replace the existing test case associated with this goal (note, however, that you still may see the replaced test in the archive if it was previously associated with other goals).
- If you assign a lower score to the new test cases than the score of your preferred test case (or their score is lower than 3), the preference archive will not be updated with them.

Therefore, be aware that <u>the preference archive will not always add a new test case for each of the interactions</u>.

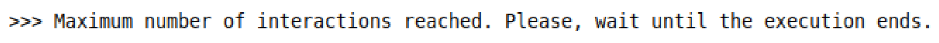
12) **Execution in progress:** The search will continue and will try to generate new test cases to cover all goals. At the beginning of the execution, interactions are more frequent; as the execution progresses, the system might require fewer interactions because it saves memory of your evaluations. Meanwhile, the system will inform you that the execution is still in progress:



```
 * The archive with your preferences has been updated and can be consulted in this folder: /home/pedro/Documentos/Example_Stack/
Execution still in progress. Please, wait until the execution ends or new interactions are required.
```

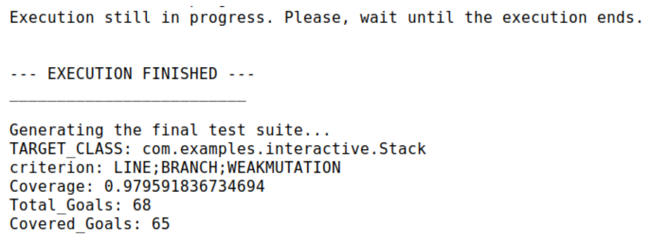
Note: Because the system saves memory of the interactions, the preference archive may be updated inadvertently during the execution, but always based on your previous evaluations.

The system will inform when the maximum number of interactions allowed is reached:



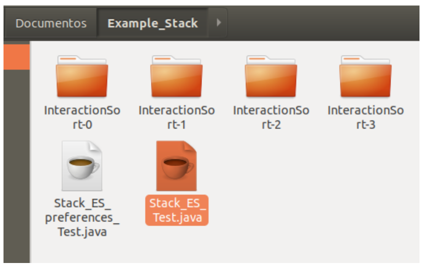
```
>>> Maximum number of interactions reached. Please, wait until the execution ends.
```

13) **Finishing the execution**: The execution will end at some point and you will see a message like this:

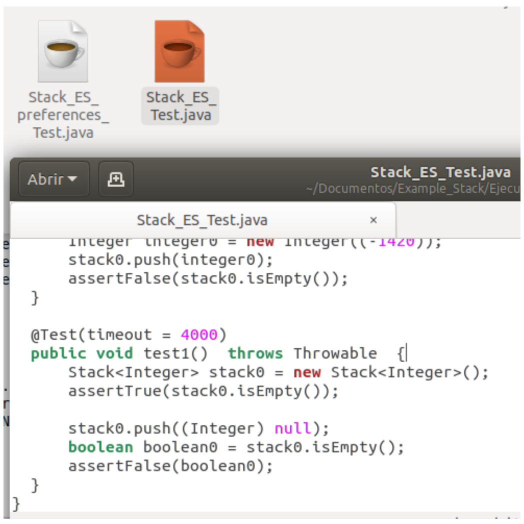


```
Execution still in progress. Please, wait until the execution ends.

--- EXECUTION FINISHED ---
_________________________

Generating the final test suite...
TARGET_CLASS: com.examples.interactive.Stack
criterion: LINE;BRANCH;WEAKMUTATION
Coverage: 0.979591836734694
Total_Goals: 68
Covered_Goals: 65
```

At that moment, you can find the resulting test suite in the directory, with the name `Stack_ES_Test.java`:



In most cases, the test cases saved in the preference archive are transferred to the resulting test suite (you will find them at the beginning of the test suite, ordered by their readability). For instance, test case number 1 is the test case that we valued the most at "Interaction 0":


Stack_ES_preferences_Test.java
Stack_ES_Test.java
Abrir
Stack_ES_Test.java
~/Documentos/Example_Stack/Ejecu
Stack_ES_Test.java

```
        Integer integer0 = new Integer((-1420));
        stack0.push(integer0);
        assertFalse(stack0.isEmpty());
    }

    @Test(timeout = 4000)
    public void test1()  throws Throwable  {
        Stack<Integer> stack0 = new Stack<Integer>();
        assertTrue(stack0.isEmpty());

        stack0.push((Integer) null);
        boolean boolean0 = stack0.isEmpty();
        assertFalse(boolean0);
    }
}
```

However, this fact does not always hold. EvoSuite performs a minimization process at the end of the search and may remove some test cases identified as *redundant* (i.e., when two test cases cover the same goal, only one of them is maintained in the final test suite).

Finally, note that, at the end of the execution, new files can appear in the same directory. They are just to collect some statistics of your interactions. Please, do not delete nor modify them.

You can run this example as many times as you want until you have understood the whole process and feel comfortable interacting with the system.