JPass Line and Branch Coverage
Verificação e Validação de Software 2020–2021

Manuel Tomás 51054
Tiago Varela 51017

2021/03/29

Contents

JaCoCo Maven Integration

The JaCoCo plugin has been integrated into the project's pom file and configured to produce a report analysing the coverage of the JUnit tests.
Specifically, the plugin has two executions, a "pre-unit-test" that sets up the necessary configurations for analysing the tests' execution and a "post-unit-test" that writes the results of the analysis into an HTML report in the "target/site/jacoco-ut" directory. Finally, an exception was added to the plugin so that jpass.ui and its subpackages are exempt from their analysis.
Furthermore, the Maven Surefire plugin, responsible for the execution of the JUnit Tests even without explicit definition in the pom file, receives the configuration set up in "pre-unit-test" as an argument line parameter, ensuring JaCoCo is executed.
This setup is based on the following tutorial: [1]

Initial Branch and Line coverage

With the tests developed in Assignment 2 and Assignment 3, the branch coverage of the all the packages (besides java.ui) was 58%, and 79% of line coverage. More specifically:

Package	Branch Coverage	Line Coverage
$jpass.util$	49%	39%
$jpass.data$	0%	0%
$jpass.xml.bind$	0%	0%
$jpass.crypt.io$[1]	86%	70%
$jpass$	0%	0%
$jpass.xml.converter$	n/a[2]	0%
$jpass.crypt$[1]	83%	100%
$Total$	58%	79%

A couple of situations are notable. The previous assignments had some failed tests.
All the ClipboardTest "[something]ExceptionTest" tests failed because the tests were incorrectly implemented, as we did not know how to force the clipboard to be busy without interfering with other tests. Furthermore, in StringUtilsTest the "shouldStripChar0x10000Test" and "shouldStripChar0x10FFFFTest" were also failing due to the char value restrictions inherent in Java. We failed to come up with a solution to any of these failures. In order to ensure accurate coverage, these tests were therefore removed (commented out).
In StringUtilsTest the "shouldStripNullTest" also failed due to the implementation not matching the documentation. As a result the implementation was corrected and the test now passes.

jpass, jpass.xml.converter, jpass.xml.bind and jpass.data are not covered by any tests. However, they are also very small packages, meaning their contribution to the instruction total is very small.

On the contrary, the jpass.crypt package is the one with the highest number of instructions and this one is completely exercised by the tests developed by the developers in combination with our own boundary testing.

The final package, jpass.util, is only 39% exercised by our tests. SpringUtilities covers the MakeGrid function well, but does not touch the MakeCompactGrid version.
ClipboardUtils exercises the functions themselves, but not the exceptions which we were originally unable to create tests for without causing concurrency issues by causing the clipboard to be busy at unexpected times. These exceptions account for around 21% of the line coverage according to JaCoCo.
CryptUtils is over half exercised by our own tests. The largest method, getSha256Hash(char[],

[1]Note that most of the coverage reported in packages $jpass.crypt$ and $jpass.crypt.io$ was done by the tests already made by the developers.

[2]This package had no branches to cover, as is has no conditions.

int), is about 3/4ths exercised. However, the segment of the method dedicated to multiple iterations of encryption is not exercised, because the public function that calls this method, getPKCS5Sha256Hash, was not tested. Finally, the method that generates a random value is called by other functions but its exception case is not exercised.
StringUtils is also over half exercised by our own tests. The largest function, stripNonValidXMLCharacters(String), is well exercised by our tests, but the other three methods are untouched.

In terms of branch coverage the branches tend to also be exercised in the segments of the source code that are well-exercised in terms of line coverage. The unexercised branches are mostly the exception cases, which, as mentioned above, often correspond to the line coverage segments that are missing.

Test cases created

To increase coverage, some tests were developed, mainly for packages we had not exercised before. Note that none of the test cases we created currently return any errors.

jpass.util.Configuration

To exercise the methods in this class, we created a ConfigurationTest file that has a test for each function.

Test	Function Exercised	What is done
shouldTestDefaultString()	get()	Attempts to get a string from the configuration file using an incorrect key. The test tests that the default value is correctly returned.
shouldTestDefaultArray()	getArray()	Attempts to get an array from the configuration file using an incorrect key. The test tests that the default value is correctly returned.
shouldTestDefaultInteger()	getInteger()	Attempts to get an Integer from the configuration file using an incorrect key. The test tests that the default value is correctly returned.
shouldTestDefaultBoolean()	is()	Attempts to get a boolean from the configuration file using an incorrect key. The test tests that the default value is correctly returned.
shouldTestDarkModeEnabled()	is()	Attempts to get the value of dark mode enabled from the configuration file using its key. The test tests that the returned value is correct.
shouldTestClearOnExitEnabled()	is()	Attempts to get the value of clear on exit from the configuration file using its key. The test tests that the returned value is correct.
shouldTestDateFormat()	get()	Attempts to get the value of the date format from the configuration file using its key. The test tests that the returned value is correct.
shouldTestDefPassLen()	getInteger()	Attempts to get the value of the default length for the generated passwords from the configuration file using its key. The test tests that the returned value is correct.

jpass.util.SpringUtilities

There were already tests for this class, but some were added to increase coverage. In particular, SpringUtilities has a method to create a Grid, but also has another equivalent one to create a Compact Grid. Therefore, we ported all of the existing Grid tests to utilise the Compact Grid method.

- **shouldNotWorkEmptyContainerCompactTest()**
  Function Exercised:
  - makeCompactGrid()

  An empty SpringLayout container is passed as an argument, which should result in an exception when attempting to create a grid with no elements there.

- **shouldNotWorkZeroAreaSizeCompactTest()**
  Function Exercised:
  - makeCompactGrid()

  A SpringLayout container with a single element is passed as an argument, and the grid area is set to zero, which should raise an exception.

- **shouldWorkPositiveAreaSizeCompactTest()**
  Function Exercised:
  - makeCompactGrid()

  Attempts to make a grid with size 1 (rows * cols == 1) which can be done as the grid will have a cell. As this is a boundary test it ensures only the boundary is not broken by checking no exception is thrown.

- **shouldNotPositiveAreaSizeGreaterThanComponentsCompactTest()**
  Function Exercised:
  - makeCompactGrid()

  Attempts to make a grid giving a Container with less elements (in the example 5) than the size of grid intended (int the example rows * cols == 6) which cannot be done has there are not enough elements to put in the grid.

- **shouldWorkPositiveAreaSizeGreaterThanComponentsCompactTest()**
  Function Exercised:
  - makeCompactGrid()

  Attempts to make a grid giving a Container with as much elements (in the example 5) as the size of grid intended (int the example rows * cols == 5) which can be done has there are enough elements to put in the grid. As this is a boundary test it ensures only the boundary is not broken by checking no exception is thrown.

Two tests failed when we initially developed them: shouldNotWorkEmptyContainerCompactTest() and shouldNotWorkZeroAreaSizeCompactTest().
Unlike MakeGrid, MakeCompactGrid was implemented in such a way that no exceptions were thrown at these unexpected values and the container was changed unduly. This was inconsistent behaviour between the two functions.
In order to ensure we could get the correct coverage we went on to fix the source code so that the tests would pass. It would be possible to either change MakeGrid's implementation to not throw an exception at the unexpected values or ensure MakeCompactGrid performed a check on the values to ensure they were valid. So as to not change implementation details, both MakeGrid and MakeCompactGrid now perform a check right at the function's onsent to ensure an exception is thrown in case these unexpected values are passed as arguments.

jpass.util.ClipboardUtils

There were already tests for this class, so to increase coverage we one test case was added to the ClipboardUtilsTest file:

- **shouldSetNullClipboardTest()**
  Function Exercised:
  - setClipboardContent()

  This test tries to set the Clipboard content with a null value, and then verifies if the current Clipboard value is indeed null.

jpass.util.CryptUtils

There were already tests for this class, so to increase coverage we one test case was added to the CryptUtilsTest file:

- **shouldGetPKCS5Sha256HashNotEmptyTest()**
  Function Exercised:
  - getPKCS5Sha256Hash()

  This test tries get the PKCS5 Sha256 Hash of a given string and verifies if it is the correct one.

jpass.util.StringUtils

There were already tests for this class, but some were added to the StringUtilsTest file:

- **shouldStripStringTest()**
  Function Exercised:

  - stripString()

  This test tries strip a string smaller than 80, which is the default value for the size striped by this method, and verifies if the string returned is the same.

- **shouldStripStringSmallerLenStringTest()**
  Function Exercised:

  - stripString()

  This test tries strip a string, using a len value smaller than the string length, and verifies if the returned string is the substring of the given one, with the size as specified by len and with "..." appended.

- **shouldStripNullStringTest()**
  Function Exercised:

  - stripString()

  This test tries strip a null string, and verifies that the returned string is null.

jpass.crypt.Cbc

There were already tests for this class that were made by the developers, so to increase coverage we created a new CbcVVSTest file to add some other test cases:

- **shouldNotDoubleDecryptSmallMessage()**
  Functions Exercised:

  - encrypt()
  - finishEncryption()
  - decrypt()
  - finishDecryption()

  This test tries to decrypt an encrypted message twice in a row without finalising the operation with another different function call, and verifies if an exception is thrown.

- **shouldNotDecryptCharOver255()**
  Functions Exercised:

  - decrypt()
  - finishDecryption()

  This test tries to decrypt string message that results in invalid values for decrypting, and verifies if an exception is thrown. (Note: It is debatable whether attempting to decrypt something that is not correctly encrypted should throw an exception, especially since the string content cannot be rightfully determined to

be encrypted or not. The exception only occurs because the format does not match the expected values, but it is essentially random whether any given String matches them.)

- **shouldNotEncryptNull()**
  Functions Exercised:
  - encrypt()

  This test tries to encrypt a null message and verifies that the encrypted message is empty.

- **shouldNotEncryptSizeNull()**
  Functions Exercised:
  - encrypt()

  This test tries to encrypt a null message and sets the encrypt method to encrypt something of length 0, then verifies that the encrypted message is empty.

- **shouldNotDecryptNull()**
  Functions Exercised:
  - decrypt()

  This test tries to decrypt a null message and verifies that that encrypted message is empty.

- **shouldNotDecryptSizeNull()**
  Functions Exercised:
  - decrypt()

  This test tries to decrypt a null message and sets the decrypt method to decrypt something of length 0, then verifies that the encrypted message is empty.

- **shouldNotEncryptNegativeSize()**
  Functions Exercised:
  - encrypt()

  This test tries to encrypt a message giving a length of -1 and verifies that that encrypted message is empty.

- **shouldNotDecryptNegativeSize()**
  Functions Exercised:
  - decrypt()

  This test tries to decrypt a message giving a length of -1 and verifies that that encrypted message is empty.

jpass.data.DataModel

To exercise the methods in this class, we created a DataModelTest file that has a test for each function.

Test	Function Exercised	What is done
shouldTestEntries()	setEntries()	A new empty "Entries" object is passed as an argument and then it is verified if the entries list is empty.
shouldTestFilename()	setFileName()	A filename is passed as an argument and then it is verified if the filename is the one given.
shouldTestModifiedFlag()	setModified()	True is passed as an argument and then it is verified if the flag has the right value.
shouldTestPassword()	setPassword()	A password is passed as an argument and then it is verified if the password is the one given.
shouldTestClear()	clear()	The method is called and then it is verified if the values returned to their default values.
shouldTestEntryByTitle()	getEntryByTitle()	The methods sets the entries list with one entry inside and then tries to get it using its title.
shouldNotTestEntryByTitle()	getEntryByTitle()	The method is called passing a non existing title and then it is verified if null is returned.

jpass.data.EntriesRepository

To exercise the methods in this class, we created a EntriesRepositoryTest file that has a test case for an unencrypted repository and an encrypted one. These aren't unit tests as they exercise the entire Entry Repository for each of the two types of repositories.

- **shouldExerciseEmptyEntriesRepositoryTest()**
  Functions Exercised:
  - newInstance()
  - writeDocument()
  - readDocument()

Initially, an instance of EntriesRepository is created using the "newInstance()" method, passing a filename as argument. Then the "writeDocument()" method is called passing an empty entries list as the argument. Finally, the method "readDocument()" is called and it is verified if the value returned is an empty entries list.

- **shouldExerciseEncryptedEmptyEntriesRepositoryTest()**
  Functions Exercised:
  - newInstance()
  - writeDocument()
  - readDocument()

  Initially, an instance of encrypted EntriesRepository is created using the "newInstance()" method, passing a filename and a key as arguments. Then the "writeDocument()" method is called passing an empty entries list as the argument. Finally, the method "readDocument()" is called and it is verified if the value returned is an empty entries list.

jpass.JPass

To cover this class we created a JPassTest file that has a unique test that exercises the main function:

- **mainTest()**
  Function Exercised:
  - main()

  This test runs the main method of the application.

Final Branch and Line coverage

With the new tests developed in this Assignment, the branch coverage of the all the packages (besides java.ui) is 90%, and 94% of line coverage. More specifically:

Package	Branch Coverage	Line Coverage
$jpass.util$	88%	83%
$jpass.data$	85%	87%
$jpass.xml.bind$	100%	48%
$jpass.crypt.io$[1]	85%	75%
$jpass$	50%	64%
$jpass.xml.converter$	n/a[2]	100%
$jpass.crypt$[1]	96%	99%
$Total$	90%	94%

There is, however, one important aspect to note: Over the course of our analysis of JaCoCo's reports as we developed the assignment we came to realise that the branch coverage as reported by JaCoCo is not the same as the one taught in class and in literature.
Consider the following example from Configuration.java in jpass.utils (line 52):
if (filePath.exists() && filePath.isFile())
JaCoCo indicates 4 branches exist in this instance. This is not how we were taught branch coverage. To apply branch coverage, two outcomes to the if should be considered: True or False. This results in two tests, not four as JaCoCo implies. For this reason, the coverage should also be 100% when both true and false are tested, but JaCoCo reports it as 50%.
Our own research later indicated a possible cause: JaCoCo analyses the Java bytecode. An example is that
if(A && B) /*instruction;*/
is converted into
if(A) if(B) /*instruction;*/
which it correctly reports as corresponding to four branches in this case. This is very similar to condition testing, but we did not confirm that it was equivalent at this time.
This has an important consequence in our testing. In order to achieve the requested 90% a lot of tests were necessary, and perhaps even more than would've been for normal branch coverage. Some of the developed tests were therefore more a result of attempts to raise the coverage rather than the result of careful analysis and test development.

References

[1] P. Kainulainen. (2013, Aug.) Creating code coverage reports for unit and integration tests with the jacoco maven plugin. [Online]. Available: https://www.petrikainulainen.net/programming/maven/creating-code-coverage-reports-for-unit-and-integration-tests-with-the-jacoco-maven-plugin/

[1]Note that most of the coverage reported in packages $jpass.crypt$ and $jpass.crypt.io$ was done by the tests already made by the developers.
[2]This package had no branches to cover, as is has no conditions.