M2GL V&V – Maven projects and test reports generation

Validation and Verification
University of Rennes 1

Erwan Bousse (erwan.bousse@irisa.fr)

Last updated September 16, 2014

Abstract

This document describes how code projects are organized with Maven in the lab sessions, and how to annotate your tests in order to automatically produce test reports.

1 IntelliJ

For all your lab sessions, you will need IntelliJ community edition `http://www.jetbrains.com/idea/download/`.

2 Maven

2.1 Importing Maven projects

For each lab session, you will download a Maven project archived in a zip file. To import it into your workspace, you must do this:

1. Extract the compressed project where you want
2. File → Open...
3. Select your project folder and validate.

The project is (almost always) completely configured with all required dependencies. Do not hesitate to be curious and to have a look in the `pom.xml` file! This is a good way to practice your Maven knowledge and training ☺.

2.2 Managing Maven goals

To be able to easily launch available Maven goals of your opened projects, go into View → Tool Windows → Maven Projects. You can then `mvn clean` or `mvn package` simply by double-clicking in this panel!

2.3 Managing JUnit tests with Maven

There are few things to know to use JUnit with Maven, but keep in mind the following:

- Maven will only look for test classes that begin with "Test", such as TestList. If you forget to put this prefix, tests will never be run by Maven.
- Tests are automatically run each time Maven compiles the project, thus for instance with the goals `package` or `compile`.

3 Generating test reports

3.1 Test descriptions with Javadoc

You will have to comment very carefully each one of your JUnit test method using javadoc tags. Except for the `@see` tag, all of them are course-specific. In fact, these annotations are configured in the `pom.xml` file of each project (see Section 2). The tags to use are described in Figure 1a. Figure 1b shows an example of how they should be used.

To generate the HTML javadoc for the code, you can use the Maven goal `javadoc:javadoc`. For the tests you can use `javadoc:test-javadoc`. Since we make links from the tests towards the code (using `@see`), we need to produce both. You can do this by double clicking on the goals in the Maven Projects panel.The output is produced in the folder `target/site/testapidocs`. Figure 2 shows an example of Javadoc run configuration.

3.2 Test coverage report with Jacoco

To generate the jacoco HTML report for the test coverage, you can use the Maven goal "`prepare-package`". Again, you can do this by double clicking on the goals in the Maven Projects panel.The output is produced in the folder `target/site/jacoco`. Figure 3 shows what the output looks like: in green the obtained coverage, in red the non-covered parts.

@see The tested method. In order for the link to work in the javadoc, use the following syntax:

`package.MyClass#methodName(type1 type2)`

@type Either “Functionnal” or “Structural”

@input The input of the method (both the calling object and the parameters)

@oracle What is expected for the test to pass

@passed Whether the test passed or not

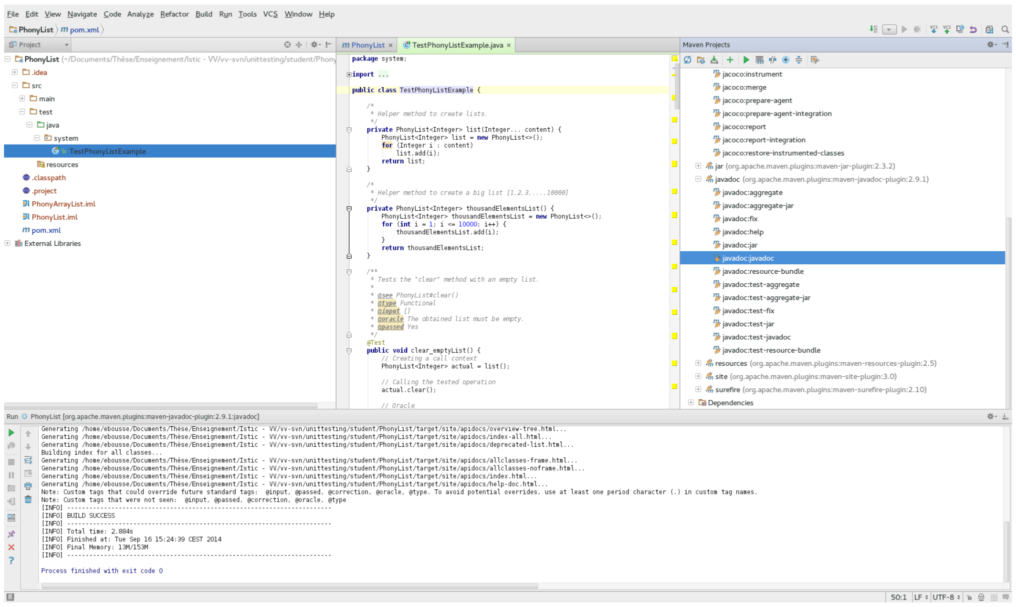
@correction If required, the applied patch. Be careful to use `<pre></pre>` to make it readable (see the example Figure 1b).

(a) List of all the annotations to use to comment the tests.

```java
public class testMyClass {

        /**
         * Tests the "doStuff" method normal behavior.
         * @see project.MyClass#doStuff(int)
         * @type Functional
         * @input 5
         * @oracle Must return "true"
         * @passed No
         * @correction
         * <pre>
         * l.9
         * - if (i > 5)
         * + if (i < 5)
         *
         * l.14
         * - value = i;
         * + value = i+2;
         * </pre>
         */
        @Test
        public void testDoStuff() {
                MyClass mc = new MyClass();
                assertFalse(mc.doStuff(5));
        }
}
```

(b) Example of JUnit test case that uses the tags.

Figure 1: Presentation of the custom Javadoc tags.

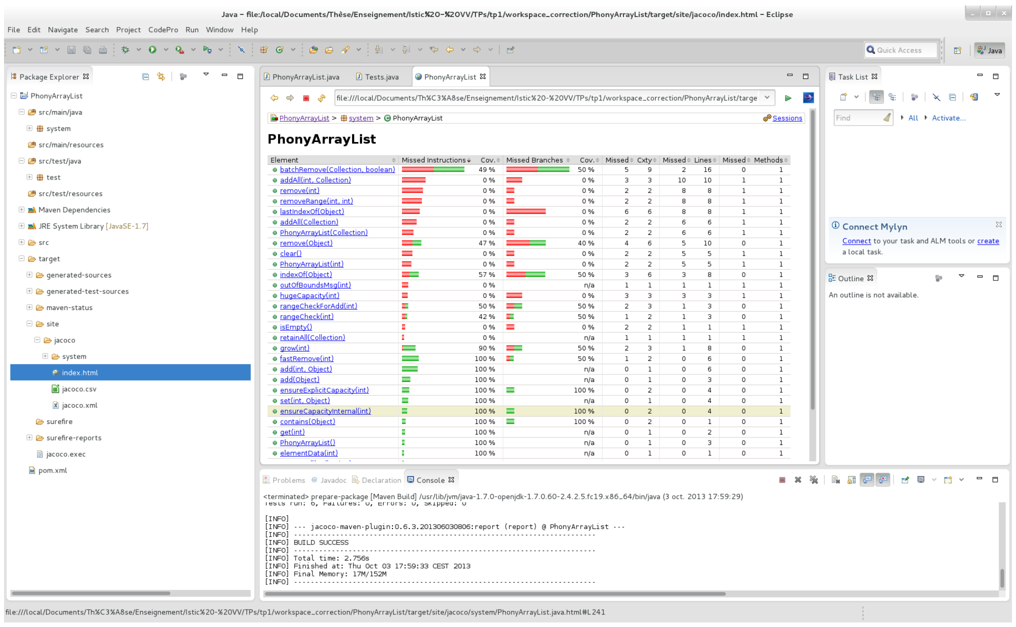


```java
package system;

import ...

public class TestPhonyListExample {

    /*
     * Helper method to create lists.
     */
    private PhonyList<Integer> list(Integer... content) {
        PhonyList<Integer> list = new PhonyList<>();
        for (Integer i : content)
            list.add(i);
        return list;
    }

    /*
     * Helper method to create a big list [1,2,3,...,10000]
     */
    private PhonyList<Integer> thousandElementsList() {
        PhonyList<Integer> thousandElementsList = new PhonyList<>();
        for (int i = 1; i <= 10000; i++) {
            thousandElementsList.add(i);
        }
        return thousandElementsList;
    }

    /**
     * Tests the "clear" method with an empty list.
     *
     * @see PhonyList#clear()
     * @type Functional
     * @input []
     * @oracle The obtained list must be empty.
     * @passed Yes
     */
    @Test
    public void clear_emptyList() {
        // Creating a call context
        PhonyList<Integer> actual = list();

        // Calling the tested operation
        actual.clear();

        // Oracle
```

Run PhonyList [org.apache.maven.plugins:maven-javadoc-plugin:2.9.1:javadoc]

```
Generating /home/ebousse/Documents/Thèse/Enseignement/Istic - VV/vv-svn/unittesting/student/PhonyList/target/site/apidocs/overview-tree.html...
Generating /home/ebousse/Documents/Thèse/Enseignement/Istic - VV/vv-svn/unittesting/student/PhonyList/target/site/apidocs/index-all.html...
Generating /home/ebousse/Documents/Thèse/Enseignement/Istic - VV/vv-svn/unittesting/student/PhonyList/target/site/apidocs/deprecated-list.html...
Building index for all classes...
Generating /home/ebousse/Documents/Thèse/Enseignement/Istic - VV/vv-svn/unittesting/student/PhonyList/target/site/apidocs/allclasses-frame.html...
Generating /home/ebousse/Documents/Thèse/Enseignement/Istic - VV/vv-svn/unittesting/student/PhonyList/target/site/apidocs/allclasses-noframe.html...
Generating /home/ebousse/Documents/Thèse/Enseignement/Istic - VV/vv-svn/unittesting/student/PhonyList/target/site/apidocs/index.html...
Generating /home/ebousse/Documents/Thèse/Enseignement/Istic - VV/vv-svn/unittesting/student/PhonyList/target/site/apidocs/help-doc.html...
Note: Custom tags that could override future standard tags:  @input, @passed, @correction, @oracle, @type. To avoid potential overrides, use at least one period character (.) in custom tag names.
Note: Custom tags that were not seen:  @input, @passed, @correction, @oracle, @type
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.884s
[INFO] Finished at: Tue Sep 16 15:24:39 CEST 2014
[INFO] Final Memory: 13M/153M
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

Figure 2: Maven window with the Maven Projects panel and the javadoc goal selected.



PhonyArrayList > system > PhonyArrayList

PhonyArrayList

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
batchRemove(Collection, boolean)		49 %		50 %	5	9	2	16	0	1
addAll(int, Collection)		0 %		0 %	3	3	10	10	1	1
remove(int)		0 %		0 %	2	2	8	8	1	1
removeRange(int, int)		0 %		0 %	2	2	8	8	1	1
lastIndexOf(Object)		0 %		0 %	6	6	8	8	1	1
addAll(Collection)		0 %		0 %	2	2	6	6	1	1
PhonyArrayList(Collection)		0 %		0 %	2	2	6	6	1	1
remove(Object)		47 %		40 %	4	6	5	10	0	1
clear()		0 %		0 %	2	2	5	5	1	1
PhonyArrayList(int)		0 %		0 %	2	2	5	5	1	1
indexOf(Object)		57 %		50 %	3	6	3	8	0	1
outOfBoundsMsg(int)		0 %		n/a	1	1	1	1	1	1
hugeCapacity(int)		0 %		0 %	3	3	3	3	1	1
rangeCheckForAdd(int)		50 %		50 %	2	3	1	3	0	1
rangeCheck(int)		42 %		50 %	1	2	1	3	0	1
isEmpty()		0 %		0 %	2	2	1	1	1	1
retainAll(Collection)		0 %		n/a	1	1	1	1	1	1
grow(int)		90 %		50 %	2	3	1	8	0	1
fastRemove(int)		100 %		50 %	1	2	0	6	0	1
add(int, Object)		100 %		n/a	0	1	0	6	0	1
add(Object)		100 %		n/a	0	1	0	3	0	1
ensureExplicitCapacity(int)		100 %		100 %	0	2	0	4	0	1
set(int, Object)		100 %		n/a	0	1	0	4	0	1
ensureCapacityInternal(int)		100 %		100 %	0	2	0	4	0	1
contains(Object)		100 %		100 %	0	2	0	1	0	1
get(int)		100 %		n/a	0	1	0	2	0	1
PhonyArrayList()		100 %		n/a	0	1	0	3	0	1
elementData(int)		100 %		n/a	0	1	0	1	0	1

```
<terminated> prepare-package [Maven Build] /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.2.5.fc19.x86_64/bin/java (3 oct. 2013 17:59:29)
Tests run: 6, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- jacoco-maven-plugin:0.6.3.201306030806:report (report) @ PhonyArrayList ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.756s
[INFO] Finished at: Thu Oct 03 17:59:33 CEST 2013
[INFO] Final Memory: 17M/152M
[INFO] ------------------------------------------------------------------------
```

file:///local/Documents/Th%C3%A8se/Enseignement/Istic%20-%20VV/TPs/tp1/workspace_correction/PhonyArrayList/target/site/jacoco/system/PhonyArrayList.java.html#L241

Figure 3: Example of jacoco report.