ETSA03-ADM-LAB4

Lab 4
Code coverage testing, static code anaysis, and Javadoc

Version 1.3 approved

Prepared by Markus Borg
Dept. of Computer Science, Lund University

April 13, 2022

Revision History

Name	Date	Reason For Changes	Version
Markus Borg	2018-04-18	Initial structure.	0.1
Markus Borg	2018-04-21	Added EclEmma, SpotBugs, and SonarLint.	0.2
Markus Borg	2018-04-22	Complete draft, ready for internal review.	0.3
Markus Borg	2018-04-24	Updated after internal review, ready for lab.	1.0
Markus Borg	2019-05-03	Updated for 2019.	1.1
Mathias Haage	2020-03-18	Minor updates for 2020.	1.2
Markus Borg	2022-04-13	Update SonarLint version.	1.3

1 Introduction

Lab 4 focuses on source code quality assurance for Basic Melee Bot. High-quality source code is your best approach to decrease the chances of releasing a robot with bugs, and it also greatly improves the maintainability of the product. First, you will add code coverage measurements to your existing suite of unit tests. Second, you will explore two established tools for static code analysis. Finally, you will generate Javadoc for Basic Melee Bot, i.e., the de facto industry standard for documenting Java classes. More specifically, Lab 4 covers the following:

- Code coverage testing with EclEmma
- Static code analysis with SpotBugs and SonarLint
- Generating Javadoc from Eclipse

2 Before the lab

You shall continue working on the source code you developed for Lab 3, but we will focus on the parts completed already during Lab 2. The source code used in Lab 4 is Basic Melee Bot and its accompanying unit test suite.

To prepare for the lab session, please make sure you can execute the unit test suite for Basic Melee Bot. Then have a look at the following pages:

- Code coverage (skip "Modified condition/decision coverage" and onwards)
  – https://en.wikipedia.org/wiki/Code_coverage
- EclEmma homepage (browse the page to get a high-level understanding)
  – http://www.eclemma.org/
- Coverage counters in JaCoCo (library used by eclEmma)
  – http://www.eclemma.org/jacoco/trunk/doc/counters.html
- Static code analysis (skip "Formal methods")
  – https://en.wikipedia.org/wiki/Static_program_analysis
- SpotBugs homepage (browse the page to get a high-level understanding)
  – https://spotbugs.github.io/
- SonarLint homepage (browse the page to get a high-level understanding)
  – https://www.sonarlint.org/
- Javadoc document generator
  – https://en.wikipedia.org/wiki/Javadoc

3 At the lab

In Lab 4, you will learn to download open source tools from the big Eclipse ecosystem through Eclipse Marketplace. You reach Eclipse Marketplace directly from your Eclipse installation: Select "Eclipse Marketplace..." from the Help menu.

3.1 Code coverage with EclEmma

First, you will work with code coverage testing. Search for EclEmma on Eclipse Marketplace, and install the latest version (unless it is already installed): EclEmma Java Code Coverage 3.1.3. EclEmma introduces a new run option in Eclipse, beyond "Run as..." and "Debug as...". The new run option is called "Coverage as...". The eclEmma "Coverage view" automatically appears when a new coverage session is added or can manually be opened from the Window → Show View menu in the Java category.

The coverage view shows code coverage summaries for the active session. Individual columns in the view contain the following numbers for the active session, always summarizing the child elements of the respective Java element:

- Coverage ratio
- Items covered
- Items not covered
- Total items

The elements may be sorted in ascending or descending order by clicking the respective column header. Double-clicking an element opens its declaration in an editor with highlighted source code. Figure 3.1 shows an example of the coverage view, and how to switch between different coverage metrics, i.e., instruction counters, branch counters, line counters, method counters, type counters, and complexity. Not that you can also remove the coverage annotation from the coverage view by clicking the cross icons.

Now it is time to perform a code coverage measurement of Basic Melee Bot to analyze to what extent the unit test suite covers the source code of the robot. In the "Package Explorer" in Eclipse, right click the unit test suite in your Java project ("BMBUnitTestSuite.java") and choose "Coverage as" → JUnit Test. The unit test suite will run as before, but you will also get coverage measurements. Explore the results in the EclEmma coverage view (skip coverage of the classes in the test package). Double click on elements in the coverage view to navigate to color coded source code, for which green = covered, yellow = partly covered, and red = not covered. Then try to answer the following questions:

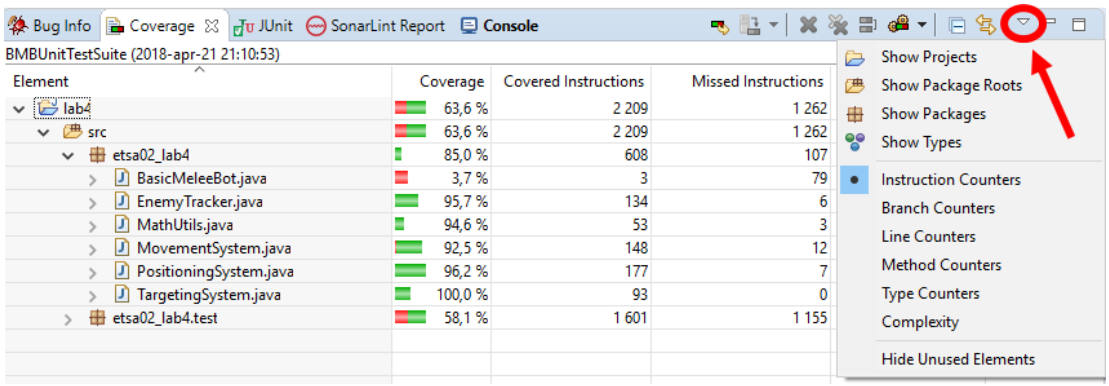

Figure 3.1: The coverage view in EclEmma. The circle shows where to switch coverage metric.

- What is the total instruction coverage?
- Which classes have the highest instruction coverage? Why?
- Which classes have the lowest instruction coverage? Why?
- What do the other coverage metrics reveal?
- Would you recommend us to add more unit tests to Basic Melee Bot?

3.2 Static code analysis with SpotBugs and SonarLint

The second part of Lab 4 is about static code analysis tools. Search for SpotBugs on Eclipse Marketplace, and install the latest version: SpotBugs Eclipse plugin 3.1.5. While you are at it, search also for SonarLint, and install the latest version: SonarLint 7.3.1.

Run SpotBugs by right clicking your Java project in Package Explorer and select SpotBugs → Find Bugs as shown in Figure 3.2. SpotBugs comes with its own Eclipse "Perspective" (a set and layout of views in the Workbench window). If the perspective has not opened yet, open it in the Window menu → Perspective → Open Perspective → Other... and select "SpotBugs" in the resulting dialog window.

In the SpotBugs perspective you have a view called "Bug Explorer", analogous to the well-known "Package Explorer". Use it to identify the issue identified by SpotBugs. Right click the issue in Bug Explorer and select "Show Bug Info" to read more. Double click the issue in Bug Explorer to directly navigate to the corresponding line in the source code. Do you understand the issue? Can you resolve it?[1] Sometimes you want to modify the severity level of issues that SpotBugs should report. To do so, right click your Java project and select Properties. Under SpotBugs you can configure the minimum rank for

[1] If you do not get any bug report, try to introduce a bug. For instance, add infinite recursion to the BasicMeleeBot.onRobotDeath method, i.e from within the method call itself (add line onRobotDeath(e);).

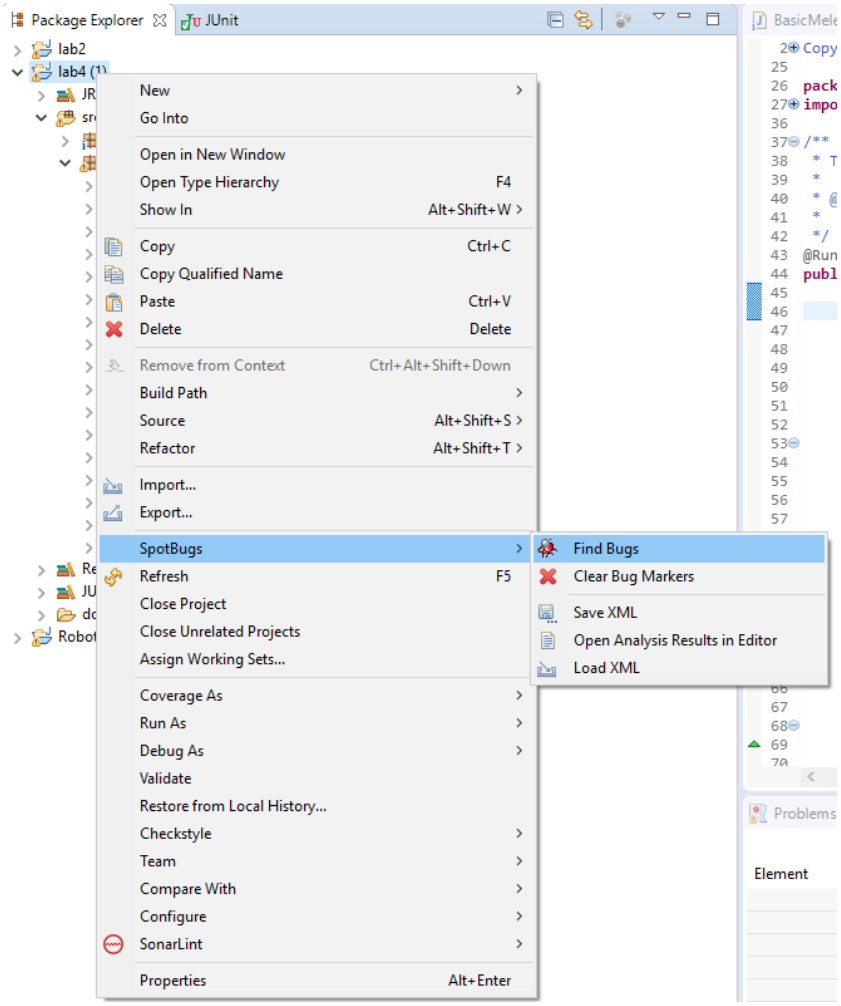

Figure 3.2: Running SpotBugs on your project.

SpotBugs to report, with severities ranging from 1 to 20 – however, the default level is already appropriate for your work in the project.

Run SonarLint by right clicking your Java project in Package Explorer and select SonarLint → Analyze. A dialog appears to warn you that it might take a while to complete this action, but this is no problem at all for projects as small as ours. SonarLint introduces new Eclipse views (like EclEmma). If they are not yet visible, open two of them in the Window menu → Show View → Other... and select "SonarLint" → "SonarLint Report" and "SonarLint Rule Description" in the resulting dialog window (cf. Figure 3.3).

Investigate the output in the SonarLint Report view. Each item represents a SonarLint rule violation, and each violation has an explicit severity level. How many items are listed? How many different types of rule violations have been identified? What severity

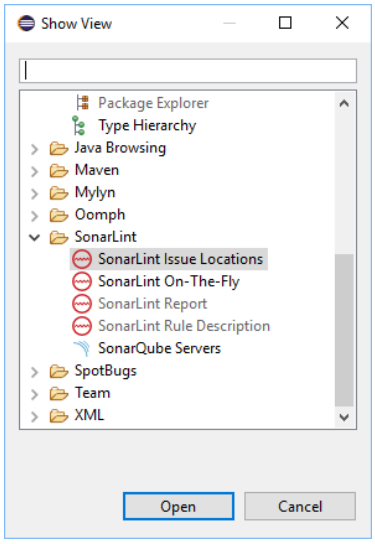

Figure 3.3: Showing the SonarLint views in Eclipse.

levels are represented in the report? You can read more about a rule violation by right clicking it and selecting "Rule description". You can also double click on any rule violation in the SonarLint Report view to navigate to the corresponding line in the source code. Can you resolve any rule violations in BasicMeleeBot to reduce the list?

3.3 Generating documentation using Javadoc

The final assignment in Lab 4 is simple, but very important for high quality software: source code documentation. The de facto standard for Java documentation is Javadoc, a tool that reads structured source code comments in Java files and outputs html formatted documentation that is easy to navigate. To generate Javadoc, select "Generate Javadoc..." in the Project menu. The dialog presented in Figure 3.4 should appear. Select the project you want documentation for, and choose to create Javadoc for all visibility levels, i.e., select Private among the first radio buttons. The default output location is fine, just proceed through the rest of the dialog windows. When the process has completed, have a look at the generated documentation by opening "index.html" in the documentation output folder. Browse the documentation. Which classes are properly documented? Which should be complemented to meet the expectations on good documentation?

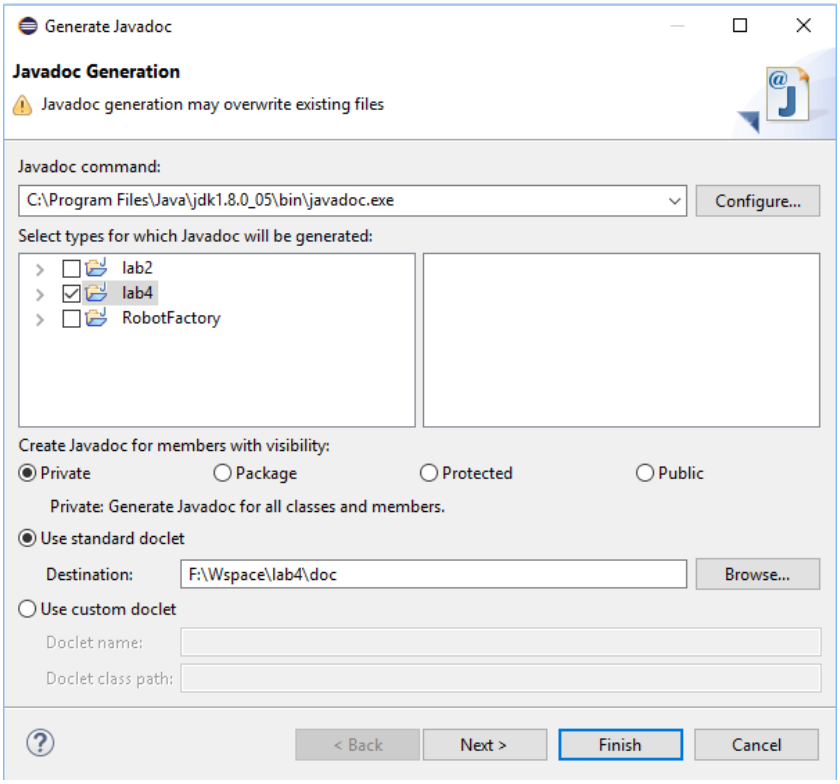

Figure 3.4: Javadoc generation in Eclipse.

4 After the lab

Your group shall state a code coverage target for your unit test suite in the STS, at least instruction coverage. For the Final Release of your robot, you shall measure code coverage using EclEmma and report the outcome in the test report. Any deviations from the coverage target shall be motivated.

In the project, your group shall run the two static code analysis tools SpotBugs and SonarLint. The latest version of the STS template specifies that you shall resolve all issues identified by SpotBugs running with the default settings, but not necessarily for the test code. Your group shall report the number of issues identified by SpotBugs in the test report that is part of the Final Release. Note that any remaining SpotBugs issues shall be explained and motivated in the test report. Your group shall also run SonarLint on your source code. As specified by the STS template, the final version of the source code shall not have any remaining SonarLint rule violations of severity major or higher. Any such remaining rule violations shall be explained and motivated in the test report.

All non-customer releases in the project shall be complemented by documentation generated by Javadoc. Make sure you provide this information for the Final Release. The class diagram and the Javadoc is what helps external stakeholders get an overview of your source code!