ETSA02-ADM-LAB2

Lab 2
Object-oriented design and unit testing

Version 1.1 approved

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

March 30, 2019

Revision History

Name	Date	Reason For Changes	Version
Markus Borg	2017-12-11	Initial draft.	0.1
Markus Borg	2018-03-21	Changed to refactoring+unit testing.	0.2
Markus Borg	2018-03-21	Complete draft.	0.3
Markus Borg	2018-03-22	Updated after internal review.	0.4
Markus Borg	2018-03-25	Clarified after more internal review.	0.5
Markus Borg	2018-03-26	Version ready for lab.	1.0
Markus Borg	2019-03-30	Updated to use the introsofteng-botexample repo.	1.1

1 Introduction

Lab 2 will inspire you to organize your robot implementation according to a a proper object-oriented design. Furthermore, the lab will help you get started with unit testing using the JUnit framework. More specifically, Lab 2 covers:

- Program comprehension, i.e., reading code of a somewhat more advanced robot.
- Reading a UML class diagram.
- Refactoring source code.
- Working with JUnit in Eclipse.

2 Before the lab

To save time during the lab session, please read through this section and follow the instructions. Also, unless you are already familiar with it, please read how the Robocode coordinate system works. One source to understand navigation in Robocode is this link: http://mark.random-article.com/weber/java/robocode/lesson4.html

Wikipedia is a great source to read up on software engineering topics. Software engineers appear to carefully monitor and describe their work practices. Perhaps they take pride in describing what they do for the masses? Good for us, Wikipedia (at least in English) is a free and often accurate source of software engineering knowledge. Besides, reading from other sources than the course book is a good idea.

Prior to the lab, please browse the following Wikipedia articles:

- Object oriented programming (skip "History" and onwards)
  – https://en.wikipedia.org/wiki/Object-oriented_programming)
- Class diagram
  – https://en.wikipedia.org/wiki/Class_diagram
- Code refactoring
  – https://en.wikipedia.org/wiki/Code_refactoring
- Unit testing
  – https://en.wikipedia.org/wiki/Unit_testing

The course material is maintained under an open source license on GitHub. Almost everything we will use in the course is stored in one of the following repositories:

- https://github.com/lunduniversity/introsofteng – This is the backbone of the course, containing the project instructions, lectures, lab instructions like this document, and exercises.
- https://github.com/lunduniversity/introsofteng-botexample – This is the skeleton of Basic Melee Bot that you will implement in Lab 2. To support you, the repository contains a suite of unit test cases. In Lab 3, you will implement automated system test cases for Basic Melee Bot. Furthermore, the repository contains all dependencies needed for the project. Your group is encouraged to use this repository as a template when creating the robot for the project.

You are all welcome to contribute to the repositories if you find errors or obvious improvements. Once you are comfortable with git, you are most welcome to create "pull requests" to improve the course material, e.g., clarifying descriptions or correcting typos!

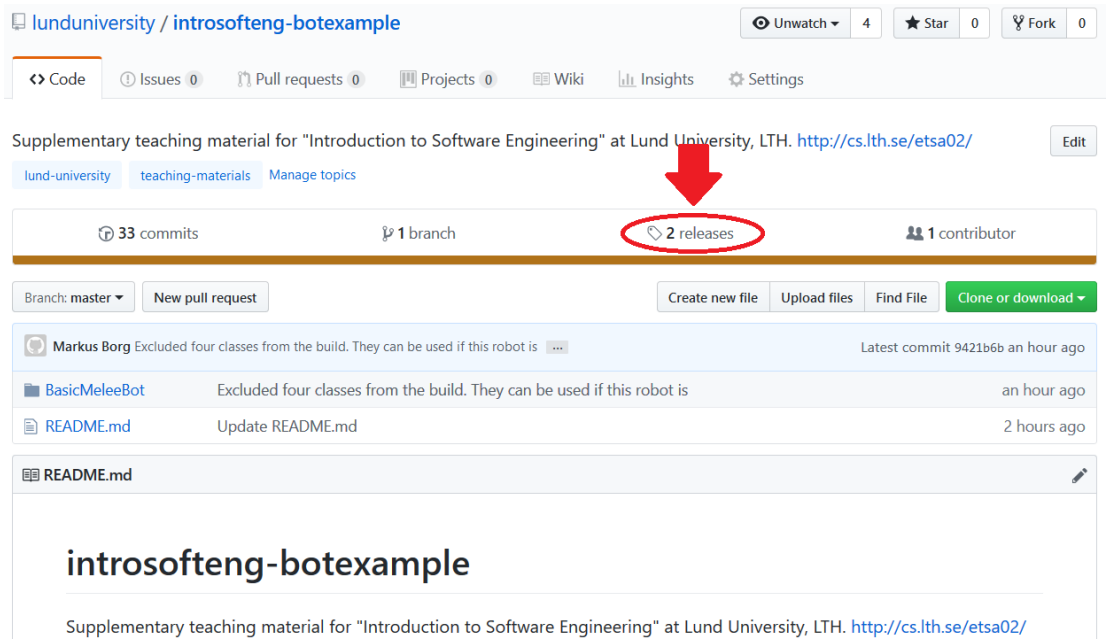

Figure 2.1: Finding the releases of introsofteng-botexample on GitHub.

To ensure that you get the correct version of the source code for Lab 2, we recommend that you download the latest *release*, i.e., a container of one or more assets, associated to a git annotated tag. GitHub uses releases to package software versions to users. Open a web browser and go to https://github.com/lunduniversity/introsofteng-botexample. Figure 2.1 shows where you find the releases created for this repository – click on it. Figure 2.2 should appear and list all available releases. For Lab 2, download the source code for v1.0.1 as a zip file as indicated by the arrow.

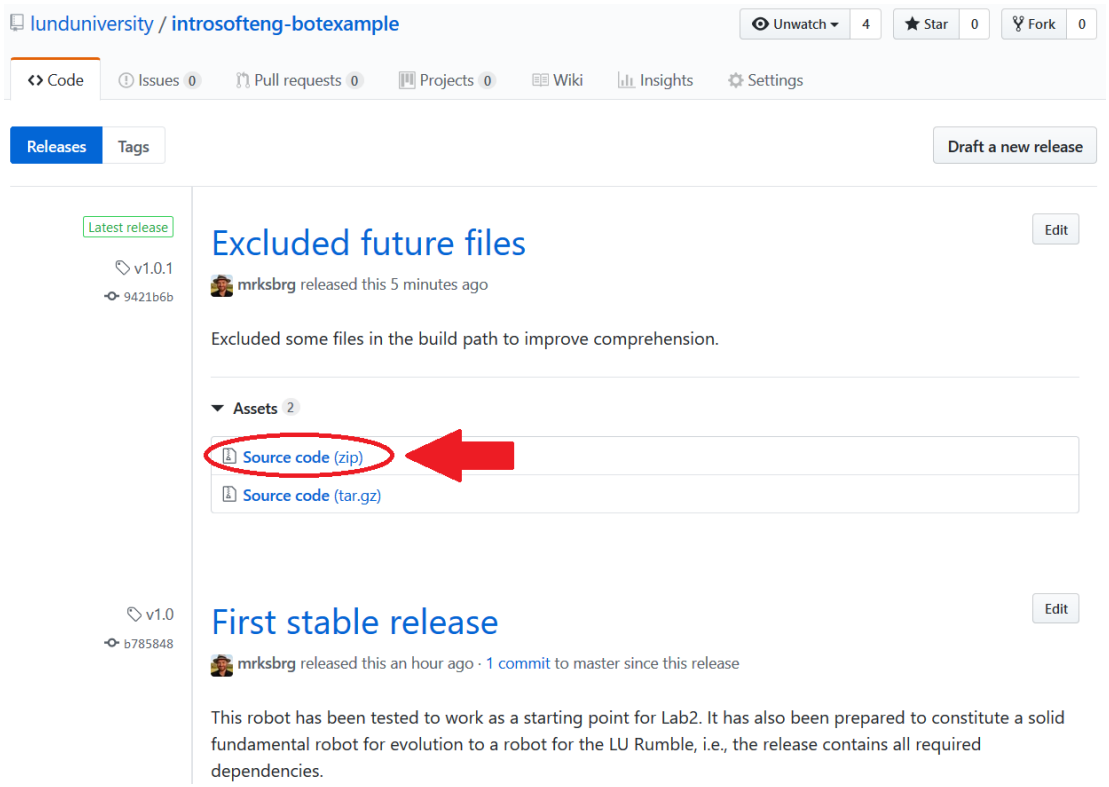

Figure 2.2: Downloading the source code corresponding to a release on GitHub. The arrow shows the zip archive.

3 At the lab

The first thing you need to do is to import the Lab 2 source code, then it is time to get started with automated testing using JUnit. Follow the steps below.

Import the source code:

1. Start Eclipse.
2. Run "Import..." from the File menu.
3. Under "General" choose "Projects from Folder or Archive" and click "Next".
4. Import the project as shown in Figure 3.1. First, click "Archive.." and select the zip file you downloaded. Second, select only the checkbox for the Eclipse project. Third, click "Finish".
5. Verify that the content in the Package Explorer resembles Figure 3.2.

Configure and run the JUnit unit test suite:

1. Right click on BMBUnitTestSuite.java in the Package Explorer. Select "Run Configurations..." under "Run As".
2. In the dialog that appears, create a new JUnit configuration. Enter the information presented in Figure 3.3.
3. Click "Run". All units tests in the test suite shall now run. All of them are supposed to fail!

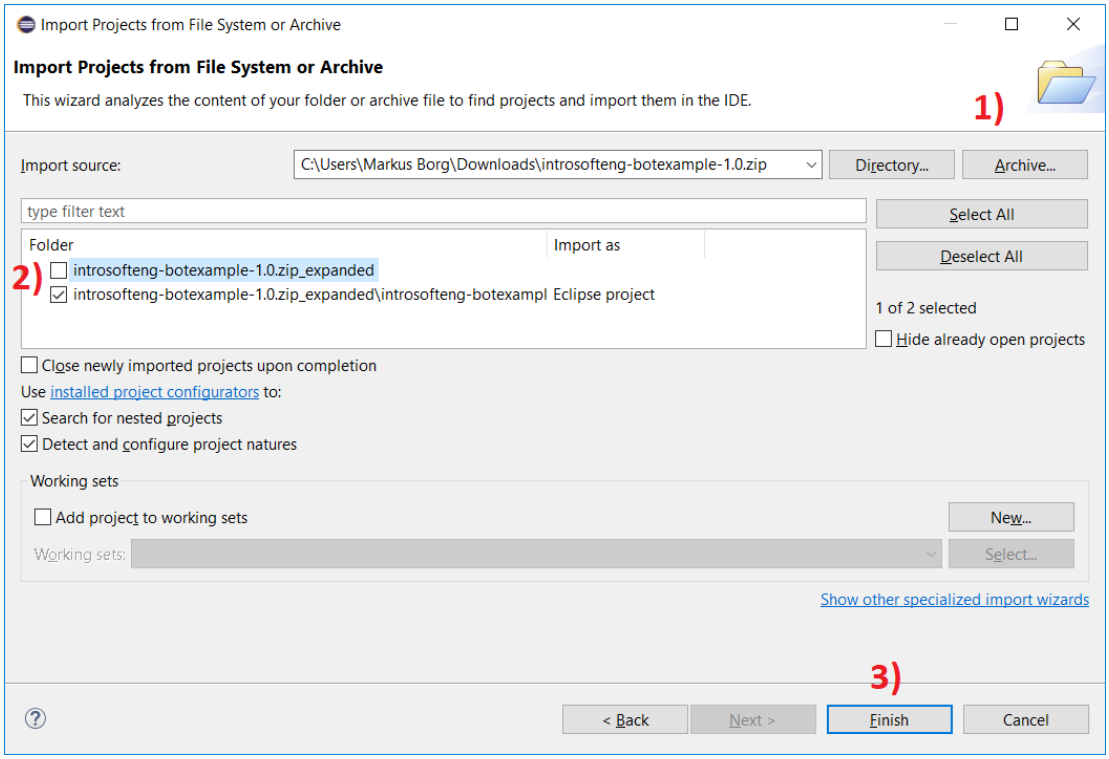

Figure 3.1: Downloading the source code corresponding to a release on GitHub. The arrow shows the zip archive.

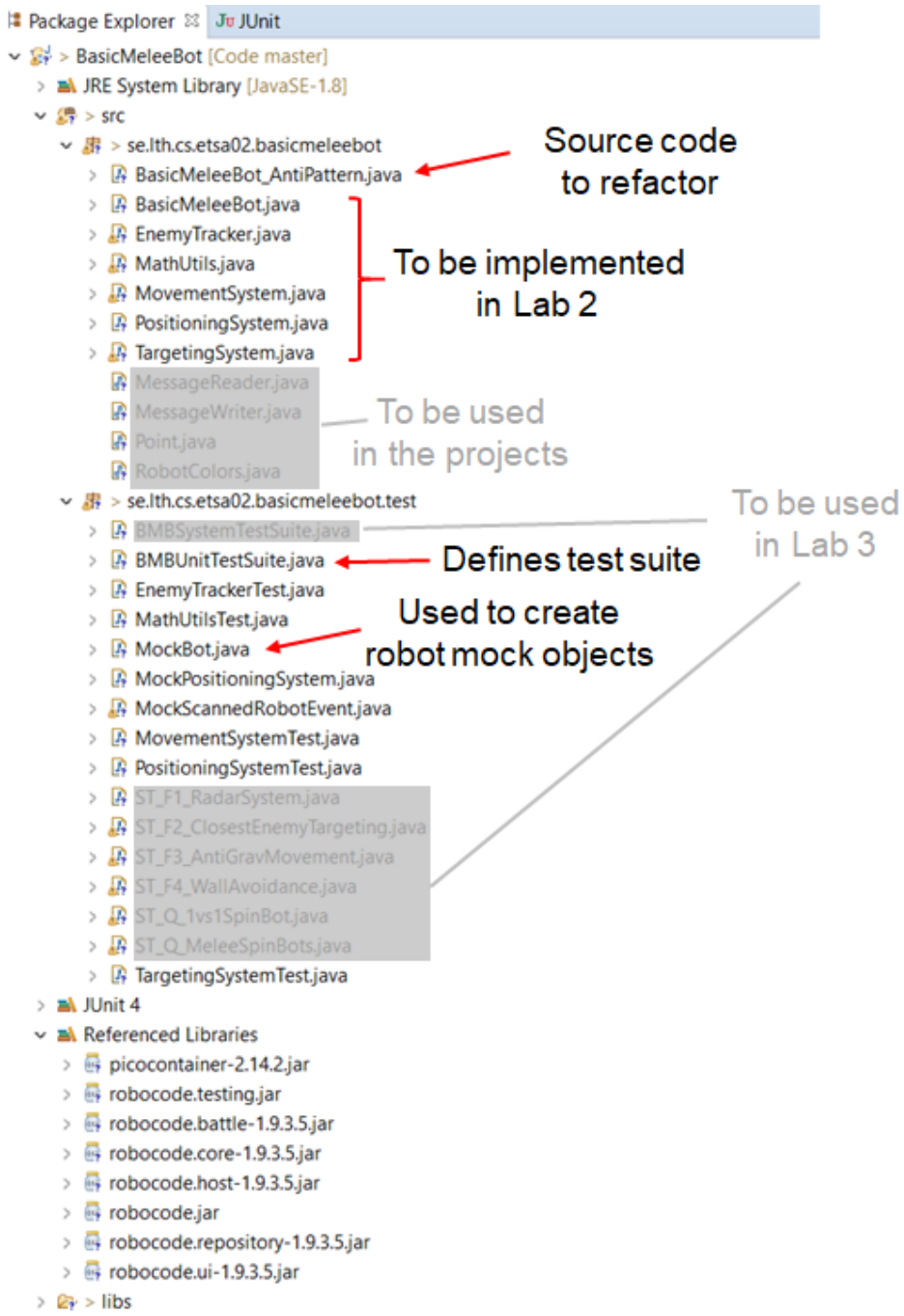

Figure 3.2: The Package Explorer in Eclipse after importing the source code from introsofteng-examplebot.

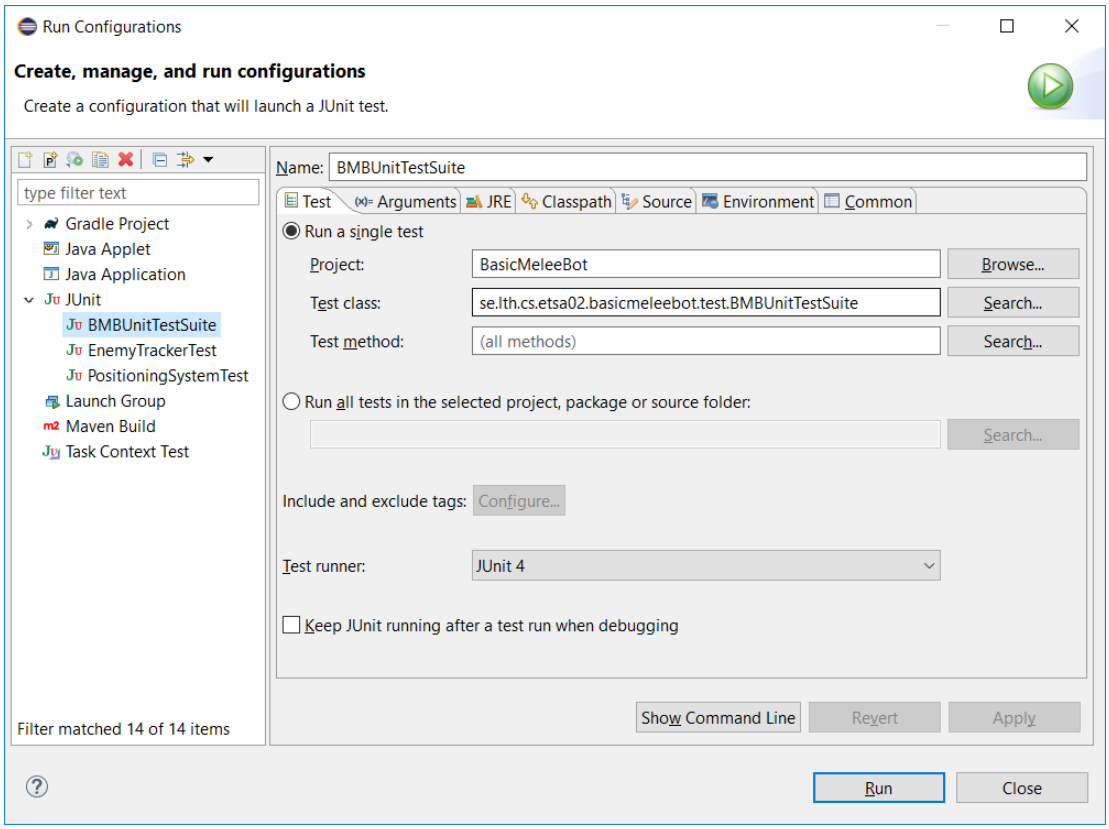

Figure 3.3: Creating a new JUnit run configuration for BMBUnitTestSuite.

Complete the refactoring task:

1. First, you should study the implementation of the robot in "BasicMeleeBot_AntiPattern". The robot implements "anti gravity movement", i.e., it moves away from other robots.

2. Try the robot in Robocode to verify the anti gravity movement. For Robocode to find Basic Melee Bot, you first need to set the path to the root directory of the BasicMeleeBot project in the "Development Options" under "Options→Preferences" in Robocode. Revisit the instructions for Lab 1 if you do not remember how to do this.

3. BasicMeleeBot_AntiPattern does not comply with good object oriented design practices, instead all functionality is collected in one big class. A modular robot design would greatly increase the evolvability of the product. Figure 3.4 shows a more feasible object oriented design. Perhaps you can use something similar in your project?

4. Your task is now to refactor BasicMeleeBot_AntiPattern into the structure presented in the provided class diagram. BasicMeleeBot.java is already implemented for you, but you should complete the missing code in the other classes. A suggested implementation order is: 1) MathUtils, 2) PositioningSystem, 3) EnemyTracker, 4) TargetingSystem, and 5) MovementSystem. The unit tests are there to guide you in the process. Study the test cases to understand the intention behind the different methods. Most importantly, run them frequently to verify that you are on the right track – when all test cases are green, you have completed Lab 2!

—— REACH THIS POINT TO PASS LAB 2! ——

5. If time permits during Lab 2, uncomment the code snippet for "wall avoidance" in MovementSystem.java and move it according to the instructions. If you did not reach this point during the lab session, please do it before Lab 3 – it is required to proceed with the automated system testing.

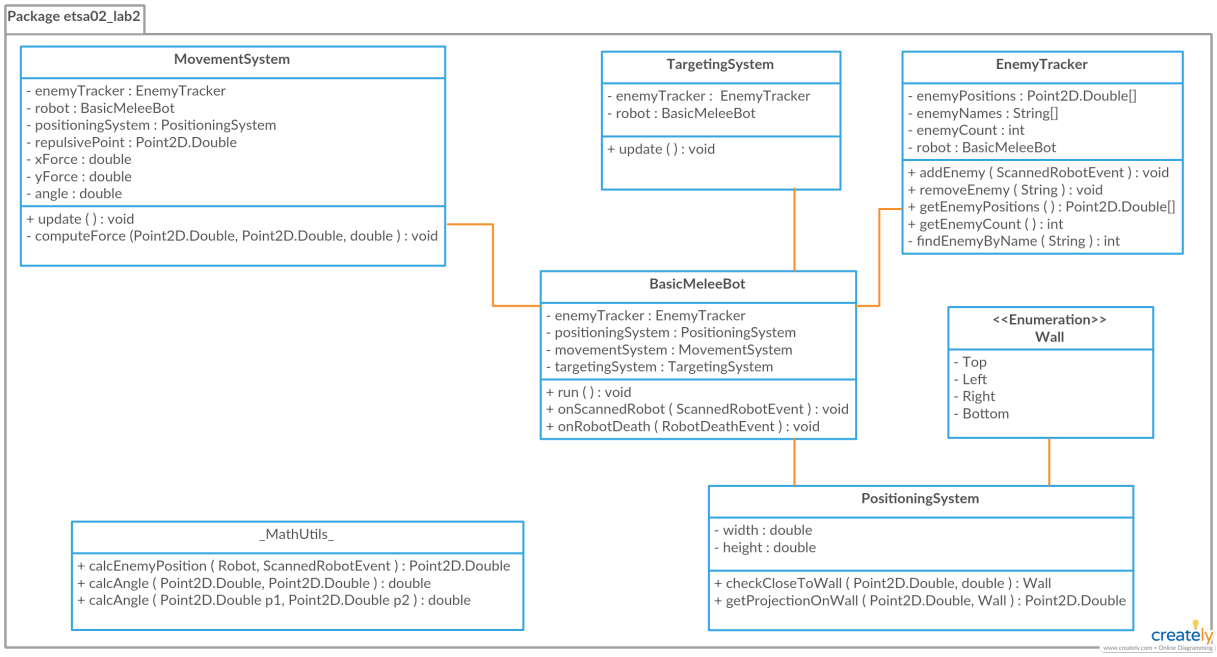

Figure 3.4: BasicMeleeBot class diagram. Note that MathUtils is a static class.

4 After the lab

During the lab you have practiced several fairly advanced software engineering practices at the same time. You have refactored source code into a modular object-oriented design, guided by a UML class diagram, while adhering to a test-driven development approach supported by automated unit testing. Well done!

When your project releases software, you are supposed to provide the regulatory body with a UML class diagram and source code complemented with a suite of JUnit test cases. Note that the group purchasing your robot shall not receive anything of this, they get only the requirements specification, the test specification, and the executable robot as a .jar file.

We recommend that the group helps the software architect to maintain a class diagram of the most recent robot design. Also, the group shall together maintain a JUnit test suite as an approach to detect bugs early – you should obviously run your test suite frequently. In industry, developers are typically responsible for writing unit tests for all their source code – this is generally a good practice. The test cases would then run every time there is a commit as part of a continuous integration pipeline.