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 29, 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	2019-03-29	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

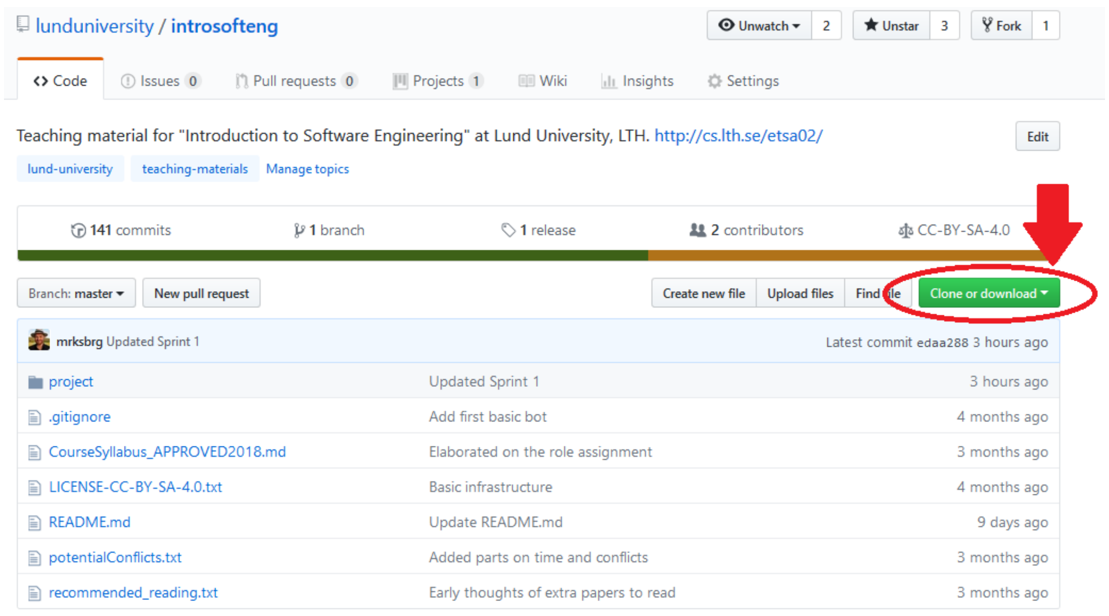
Wikipedia is a great source to read up on software engineering topics. Software engineers appear to carefully monitor and describe their work practices. Perhaps they take pride in describing what they do for the masses? Good for us, Wikipedia (at least in English) is a free and often accurate source 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. Have a look at: https://github.com/lunduniversity/introsofteng If you are already familiar with git, I strongly recommend that you clone the repository. If you don't know what this means, instead click the button presented

TODO: RERWITE in Figure 2.1 and choose "Download ZIP". Your lab instructor can help you getting started with git, but simply extracting the Lab2 files from the zip archive works fine as well. For Lab 2, you need the files located in introsofteng-master/project/rumble/labs/lab2/src, including its subfolder: 'test'. Watch the video "Lab2_download.avi" on Google Drive (ETSA02 Everyone/Labs) if you need support. 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!



Teaching material for "Introduction to Software Engineering" at Lund University, LTH. http://cs.lth.se/etsa02/

mrksbrg Updated Sprint 1 — Latest commit edaa288 3 hours ago

project	Updated Sprint 1	3 hours ago
.gitignore	Add first basic bot	4 months ago
CourseSyllabus_APPROVED2018.md	Elaborated on the role assignment	3 months ago
LICENSE-CC-BY-SA-4.0.txt	Basic infrastructure	4 months ago
README.md	Update README.md	9 days ago
potentialConflicts.txt	Added parts on time and conflicts	3 months ago
recommended_reading.txt	Early thoughts of extra papers to read	3 months ago

Figure 2.1: TODO: UPDATE introsofteng repository on GitHub. The arrow shows the "Clone or download" button.

3 At the lab

The first thing you need to do is to get started with automated testing using JUnit. Start by following the steps below. If you need more detailed instructions, watch the videos “Lab2_createAndImport.avi” to create a Java project and import the required files and “Lab2_runJUnit.avi” to create a JUnit run configuration. You can download both videos from Google Drive (ETSA02 Everyone/Labs).

1. **Create a new Java project for Lab2 in your Eclipse workplace.** As in Lab 1, add the robocode.jar as an “External JAR” in the Libraries tab, either when creating the project or afterwards in the project properties.

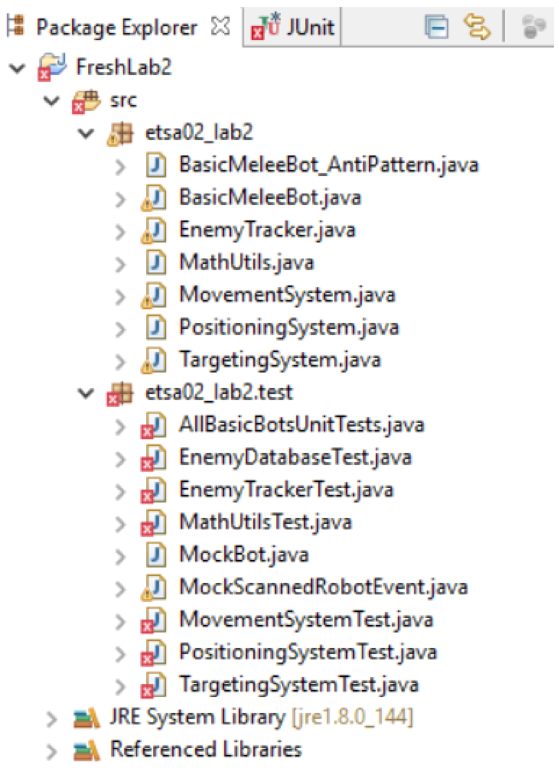
2. **Create a new package called etsa02_lab2**. In the “Package Explorer” pane, right click “src” and create a new “Package” called “etsa02_lab2”.

3. **Import the source files required for lab2**, either from the cloned repository or the zip archive. In the “Package Explorer” pane, right click the newly created package and run “Import...”. Click “General” and then choose “File system” and select all downloaded files in both the “src” folder and the “src/test” folder. The content in the Package Explorer should resemble Figure 3.1.

4. **Add JUnit as a library**. To make the etsa02_lab2.test package build, you must configure JUnit. Right click the Java project and select “Build Path” and click “Add Libraries...” and select JUnit. Click “Next >” followed by “Finish” to accept the default settings. You should now see JUnit in the Package Explorer in Eclipse. (In the video, I did this at the same time as I added the robocode.jar)

5. **Create a JUnit run configuration**. Right click “AllBasicMeleeBotUnitTests.java” and choose “Run as..” and create a new JUnit run configuration. Give the configuration a descriptive name, such as “Lab2 unit tests”.

6. **Perform unit testing**. Run the unit tests. All of them are supposed to fail!



```
FreshLab2
  src
    etsa02_lab2
      BasicMeleeBot_AntiPattern.java
      BasicMeleeBot.java
      EnemyTracker.java
      MathUtils.java
      MovementSystem.java
      PositioningSystem.java
      TargetingSystem.java
    etsa02_lab2.test
      AllBasicBotsUnitTests.java
      EnemyDatabaseTest.java
      EnemyTrackerTest.java
      MathUtilsTest.java
      MockBot.java
      MockScannedRobotEvent.java
      MovementSystemTest.java
      PositioningSystemTest.java
      TargetingSystemTest.java
  JRE System Library [jre1.8.0_144]
  Referenced Libraries
```

Figure 3.1: The Package Explorer in Eclipse after importing the Lab 2 source code.

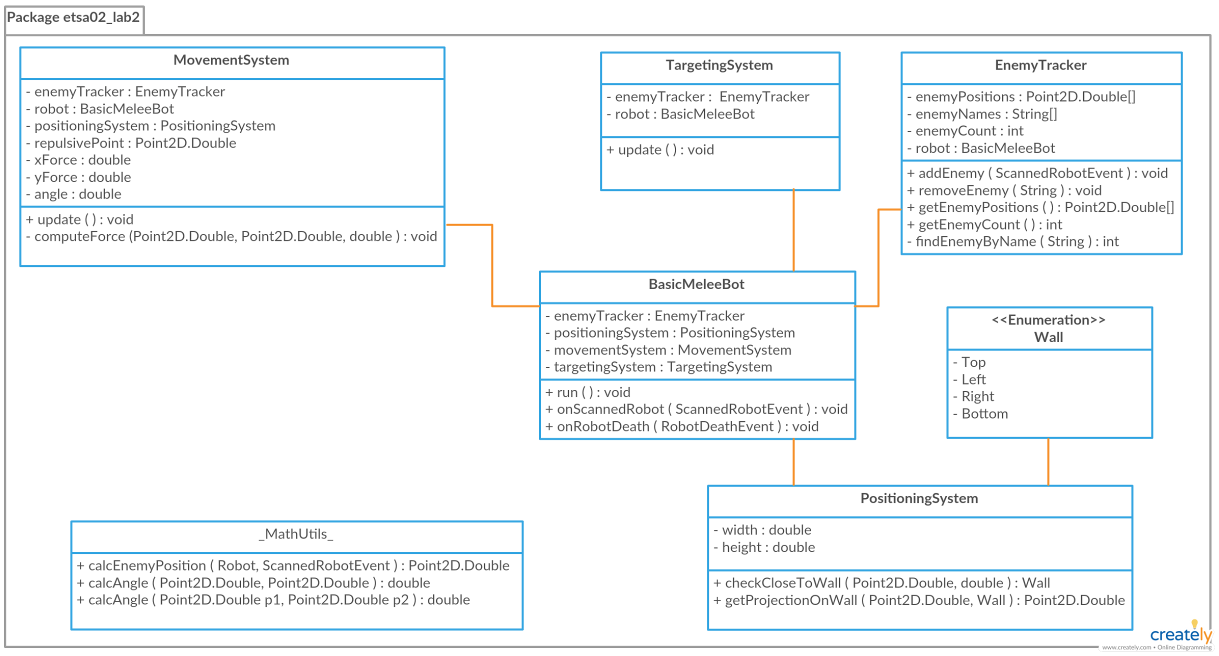

Package etsa02_lab2
MovementSystem
- enemyTracker : EnemyTracker
- robot : BasicMeleeBot
- positioningSystem : PositioningSystem
- repulsivePoint : Point2D.Double
- xForce : double
- yForce : double
- angle : double
+ update ( ) : void
- computeForce (Point2D.Double, Point2D.Double, double) : void
TargetingSystem
- enemyTracker : EnemyTracker
- robot : BasicMeleeBot
+ update ( ) : void
EnemyTracker
- enemyPositions : Point2D.Double[]
- enemyNames : String[]
- enemyCount : int
- robot : BasicMeleeBot
+ addEnemy (ScannedRobotEvent) : void
+ removeEnemy (String) : void
+ getEnemyPositions ( ) : Point2D.Double[]
+ getEnemyCount ( ) : int
- findEnemyByName (String) : int
BasicMeleeBot
- enemyTracker : EnemyTracker
- positioningSystem : PositioningSystem
- movementSystem : MovementSystem
- targetingSystem : TargetingSystem
+ run ( ) : void
+ onScannedRobot (ScannedRobotEvent) : void
+ onRobotDeath (RobotDeathEvent) : void
<<Enumeration>>
Wall
- Top
- Left
- Right
- Bottom
PositioningSystem
- width : double
- height : double
+ checkCloseToWall (Point2D.Double, double) : Wall
+ getProjectionOnWall (Point2D.Double, Wall) : Point2D.Double
MathUtils
+ calcEnemyPosition (Robot, ScannedRobotEvent) : Point2D.Double
+ calcAngle (Point2D.Double, Point2D.Double) : double
+ calcAngle (Point2D.Double p1, Point2D.Double p2) : double
creately

Figure 3.2: BasicMeleeBot class diagram. Note that MathUtils is a static class.

Now it is time to start 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.

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 3.2 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 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!

The following instructions are optional, in case you finish the refactoring task quickly. If you complete this task, you will have something similar to the BasicMeleeBot that will be offered as an ETSA02 basic bot on Robot Market. If you don't complete this task during the lab session, you will anyway get the corresponding source code.

Try to improve the behavior of BasicMeleeBot. As you might have noticed, the robot gets stuck in walls when it performs the anti gravity movement. A standard approach to solve this is to implement some form of "wall avoidance" strategy. Add wall avoidance to the update() method in the class MovementSystem, and use the methods in PositioningSystem to implement this improvement. Is it possible to add any unit tests for this behavior? If so, add new unit tests to the test suite in MovementSystemTest. If not, try to explain why.

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 should 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 should 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.