Problem Set 1 | Problem Set 1 Beta | Contenu du cours 6.005.1x

The purpose of this problem set is to:

- introduce the tools we use in 6.005x, including Java, Eclipse, and JUnit;
- introduce the process for 6.005x problem sets;
- and practice reading a function specification and writing basic Java to implement it.

You should focus in this problem set on writing code that is safe from bugs and easy to understand, using the techniques we discuss in class. You won't be asked to write any test cases of your own on this problem set. Testing will happen in Problem Set 2.

Configure Eclipse

Eclipse is a powerful, flexible, and occasionally frustrating set of tools for developing and debugging programs, especially in Java.

> *Note: if you prefer, you can do the problem sets in another Java development environment, or using command-line tools. The problem sets require compiling Java, running JUnit, and running Ant, so if you know how to do those things some other way, go for it. The rest of these instructions will talk only about using Eclipse.*

You should have already installed Eclipse and Java when you set up the Java Tutor.

When you run Eclipse, you will be prompted for a "workspace" directory, where Eclipse will store its configuration and metadata. The default location is a directory called `workspace` in your home directory. You should not run more than one copy of Eclipse at the same time with the same workspace.

On the left side of your Eclipse window is the Package Explorer, which shows you all the projects in your workspace.

1. Open Eclipse preferences.

   Windows & Linux: go to *Window → Preferences*.

   OS X: go to *Eclipse → Preferences*.

2. Make sure Eclipse is configured to use **Java 8**.

   1. In preferences, go to *Java → Installed JREs*. Ensure that "Java SE 8" or "JDK 1.8.0_71" is the only one checked. If it's not listed, click Search.

   2. Go to *Java → Compiler* and set "Compiler compliance level" to 1.8. Click OK and Yes on any prompts.

3. Make sure **assertions are always on**. Assertions are a great tool for keeping your code safe from bugs, but Java has them off by default.

   In preferences, go to *Java → Installed JREs*. Click "Java SE 8", click "Edit…", and in the "Default VM arguments" box enter: `-ea` (which stands for *enable assertions*).

4. Make sure your **Eclipse workspace refreshes automatically** whenever files are changed on the filesystem. This will be important when you run the grader script, so that you see the output files it generates.

   In preferences, go to *General → Workspace*. Turn on both checkboxes that talk about refresh: "Refresh using native hooks or polling" and "Refresh on access."

5. **Tabs**. If you configure the editor to use spaces instead of tabs, then your code will look the same in all editors regardless of how that editor displays tab characters.

   In preferences, go to *Java → Code Style → Formatter*. Click the "Edit…" button next to the active profile. In the new window, change the Tab policy to "Spaces only." Keep the Indentation size and Tab size at 4. To save your changes, enter a new "Profile name" at the top of the window and click OK.

The 6.005 Eclipse FAQ has some tips and tricks to help you make the most of Eclipse.

Download and Import the Problem Set Code

1. Download the starting code for this problem set:

   > *ps1.zip*

2. Import the code into Eclipse:

   1. In the Eclipse menubar, choose File → Import...
   2. In the Select dialog, open General and choose Existing Projects Into Workspace.
   3. In the Import Projects dialog, choose Select archive file, then click Browse and use the file dialog to find where you downloaded ps1.zip.
   4. Still in the Import Projects dialog, make sure ps1-warmup is listed in the Projects, and that the checkbox next to it is checked
   5. On "Import projects," make sure ps1-warmup is checked.

      > *If ps1-warmup is grayed and uncheckable, then the likely reason is that ps1-warmup already exists in your Package Explorer (perhaps because you are following these directions a second time). You need to cancel the import, delete or rename ps1-warmup in your Package Explorer, and try the import again.*

   6. Click Finish. ps1-warmup should now appear in your Package Explorer.

Warm up with `Quadratic.roots()`

In the Eclipse Package Explorer, open up `ps1-warmup / src / ps1`, and you should find the file `Quadratic.java`. Your warm-up task is to implement the `roots()` method in this class, which finds the roots of a quadratic equation. The method's specification comment describes how it should behave. Pay close attention to this specification while you're implementing. The quadratic formula will be useful in your solution, but be careful! Many things can go wrong.

Before you start writing code, read the rest of this handout, because it describes three ways that you can use to

check your implementation: (1) a `main` method, (2) a JUnit test suite, and (3) an autograder. Read about and try running each of these first, using the unfinished `roots()` implementation that we provided you, and then come back and start working on your implementation.

Running `main()`

The `Quadratic` class also contains a `main` method. The method `public static void main(String[] args)` is the standard entry point for a Java program. In this case, the `main` method calls `roots()` on a simple quadratic equation.

To run `main`, right-click on `Quadratic.java` in the Package Explorer, and choose *Run As → Java Application*. You will see the results printed in the Console tab at the bottom of the Eclipse window. It will say "not implemented yet" if you haven't started writing `roots()`

Run JUnit Tests

A `main` method is not a great way to test a program. Much better is automated unit testing, which runs a suite of tests to automatically test whether the implementations are correct.

JUnit is a widely-adopted Java unit testing library, and we will use it heavily in 6.005.

To find the tests for Quadratic, open up `ps1-warmup/test/ps1` in the Package Explorer, where you should find the file `QuadraticTest.java`. By convention, JUnit tests are put in a class whose name ends in Test, and 6.005 goes a step further by putting them in the `test/` folder rather than the `src/` folder.

To run the tests, right click on `QuadraticTest.java` in the Package Explorer, and choose *Run As → JUnit Test*. You should see the JUnit view appear.

If your implementation of `roots()` is correct, you should see a green bar, indicating that all the tests passed.

If your implementation is still broken or unfinished, you should see a red bar, and a list of the tests that were run. Clicking on a test shows a *stack trace* in the bottom box, which provides a brief explanation of what went wrong. Double-clicking on a line in the stack trace goes to the code for that frame in the trace. This is most useful for lines that correspond to your code; this stack trace will also contain lines for Java libraries or JUnit itself.

Use these tests to help you implement roots().

Run the Autograder

Once your implementation is passing all the JUnit tests, you need to run the automatic grader. The autograder is the file `grader.xml` in your project. It is written as an Ant script, where Ant is an example of a *build management tool*, a kind of tool commonly used in software development to compile code, run tests, and package up code for deployment. Our autograder script does all three of these things. Support for running Ant scripts is built into Eclipse.

To run the automatic grader, right click on `grader.xml` in the Package Explorer, and choose *Run As → Ant Build*. Choose the plain *Ant Build* rather than *Ant Build...*, because you don't need all the options that the Ant Build... dialog box will offer you.

The grading script will run and display output in the Console tab. At the end, it should tell you that it has created two new files: `my-grader-report.xml`, which is the result of running the grading tests, and `my-submission.zip`, which is your problem set code and grader output packaged up and ready for submission to edX.

> *Note: if automatic workspace refreshing is not working for some reason, then `my-grader-report.xml` and `my-submission.zip` may not immediately appear in the Eclipse Package Explorer. Try refreshing the project manually by right-clicking on `ps1-warmup` and choosing Refresh. Then you should see the files appear. If they still aren't there, check the Console tab for errors.*

To view your autograder results, double-click on `my-grader-report.xml`, and you will see the results in your JUnit pane. The `my-grader-report.xml` file is simply the output of running JUnit on the autograder's tests. For now (Problem Set Beta 1), the autograder tests are identical to the tests you've already been using in QuadraticTest.java, so you should see exactly the same tests passing or failing that you see when you ran JUnit yourself as in the previous section. On future problem sets, the autograder will run different tests.

You can change your code and rerun the autograder as often as you want. You have to double-click on `my-grader-report.xml` each time to see the newest results. You don't need to Refresh the project each time, however. Once `my-grader-report.xml` is visible in the Package Explorer, Eclipse will load the latest version whenever you click on it.

Submit the Beta version of your Problem Set

After the autograder runs, it produces `my-submission.zip` in your Eclipse project. This zip file contains your code and your grading report. To get credit for the problem set, you need to upload this zip file to edX before the deadline.

The Problem Set Beta 1 submission page is the last section of this handout. To find it, click the rightmost button on the section bar at the top of this page: