CS2110 Fall 2015 Assignment A2: Writing functions in Java

Introduction

Last fall, we asked the 2110 students in the first few weeks to become familiar with Java Strings, if-statements, etc. Many students did not take this seriously, and seven weeks later, we found that many were still very shaky with these features. We don't want this to happen again, so we give this little assignment now. A2 is due *after* assignment A1 is due; we hand it out now because some want it. We hand out A1 at the third lecture of the course.

Please keep track of the time you spend on A2. You will have to tell us when you submit A2.

Learning objectives

- Gain familiarity with String methods, if-statements, assignment, functions, and begin learning about Java loops.
- Get used to Eclipse and JUnit testing.

Collaboration policy

You may do this assignment with one other person. If you are going to work together, then, as soon as possible —and certainly before you submit the assignment— get on the CMS for the course and do what is required to form a group. Both people must do something before the group is formed: one proposes, the other accepts. If you need help with the CMS, visit www.cs.cornell.edu/Projects/CMS/userdoc/.

If you do this assignment with another person, *you must work together*. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. You should take turns "driving" —using the keyboard and mouse.

Academic Integrity

With the exception of your CMS-registered partner, you may not look at anyone else's code from this semester or a previous semester, in any form, or show your code to anyone else, in any form. You may not show or give your code to another person in the class.

Getting help

If you don't know where to start, if you don't understand testing, if you are lost, etc., please SEE SOMEONE IMMEDIATELY —a course instructor, a TA, a consultant, the Piazza for the course. Do not wait.

What to do

File `A2.java`, on the course website on the assignments page, contains 6-7 functions for you to write. Instructions are given at the top of that file and also within the body of each function, as comments. Complete the functions and submit completed, correct file `A2.java` on the CMS by the end of the due date given on the CMS.

Your grade will depend only on the correctness of the functions, determined by running some test cases on each function. If a function is incorrect, you may receive 0 for it. We now explain how you can get all functions correct.

1. Use a JUnit test class and test each function thoroughly, as you write it. We explain how to use a JUnit test class on the next page.
2. Well before the deadline for A2, we will make our suite of test cases available, so you can check your functions against our test cases. That way, you can ensure before submitting that your functions are correct.

You can skip step 1. This has two disadvantages. First, you don't get practice in coming up with your own test cases and with JUnit testing. Second you have to wait before getting started on the assignment.

When your functions pass all our tests, so you know A2 is correct, *fill in the appropriate line at the top of A2 with niters and hours and minutes* and submit A2 on the CMS. Everyone should get 100 on this assignment.

Guidelines/Instructions for working with Eclipse and JUnit testing in A2

1. Create a new project `a2` (or any name you want), using menu item: `File -> New -> Java Project`.

2. In the Package Explorer, click the right arrow preceding the project name. It turns into a down arrow and you see a directory `src` (for source) with nothing in it.

3. Drag the downloaded file `A2.java` on top of `src`. A window will pop up, asking whether to link or copy. Select *copy* and click `OK`.

4. There is now a right arrow before `src`. Click it. It turns into a down arrow and under it is a right arrow followed by `(default package)`. Click that right arrow. It turns into a down arrow and under it you see `A2.java`. It contains a copy of the file that you dragged.

5. Double click file name `A2.java`. It opens in the big pane to the right of the Package Explorer pane. You can now edit that file.

We now give instructions for creating and using a JUnit testing class.

6. Make sure `A2.java` is selected in the Package Explorer pane.

7. Use menu item `File -> New JUnit Test Case`. In the window that opens, make sure the "Name:" field is `A2Test`. Click `Finish`. You will be asked whether to add JUnit 4 to the build path. Yes you do. Click `OK`.

8. You can see that file `A2Test.java` has been added to the default package of the project, and it appears in the editing pane. It has a static procedure test.

9. Copy and paste the following code into file `A2Test.java`, after procedure `test` and before the end `}`:

```
@Test
public void testSumDif() {    // assertEquals(expected value, computed value);
  assertEquals(8, A2.sumDif(true, 5, 3));
  assertEquals(2, A2.sumDif(false, 5, 3));
  assertEquals(0, A2.sumDif(true, 5, 3));
}
```

A call on procedure `assertEquals` has two arguments. The second is a value that you want to compute and test, and the first is the expected value of that computation. If they are not the same, execution of the procedure will terminate abnormally (an exception will occur). Above, the first two `assertEquals` calls will terminate normally; the third should abort, since the sum of 5 and 3 is not 0.

10. With file `A2Test.java` selected in the Package Explorer and file `A2Test.java` open in the editing pane, use menu item `Run -> Run`. This executes calls on `test` and `testSumDif`. Now, instead of the Package Explorer pane on the left, you see a JUnit pane, with a red line, meaning that some test procedure aborted, i.e. threw an exception. Under it, you see an arrow followed by `A2Test`. If it is a right arrow, click on it to reveal its contents. Now, the two blue x's for `test` and `testSumDif` shows that both of them aborted.

11. In procedure `test`, execution of the call on procedure `fail` *always* fails. Double-click on `testSumDif` in the JUnit pane, causing the third `assertEquals` call to be highlighted, because that caused the exception. Look at the `Failure Trace` in the bottom of the JUnit pane. It says that 0 was expected but it was 8.

12. Put `//` before the annotation `@Test` before procedure `test`, thus commenting out the annotation. In the same way, make the third `assertEquals` call into a comment. Then do menu item `Run -> Run` again.

13. You now see a green line —no exception occurred. Click the right arrow before `A2Test` in the JUnit test pane. You see that only *one* procedure call was executed. A call on procedure test was *not* executed because it was not preceded by the annotation `@Test`.