CMPSCI 187 / Spring 2015
HamSpam

Due on 2015-01-29 08:30 EST

Marc Liberatore and John Ridgway

Morrill I N375
Section 01 @ 10:00
Section 02 @ 08:30

Contents

Overview

For this assignment, you will need to install and configure Eclipse. You will learn the standard procedure that you will use in this class for completing a programming assignment, testing your code, and turning it in through Moodle.

In this assignment and others, you will be starting from some initial code provided by us and then modifying it. Since you will repeat a similar process for each programming assignment, make sure you can complete the steps in this assignment and that you understand each one.

We strongly suggest you start this assignment early so that you can fix any Eclipse- or Moodle-related problems well before the deadline. Remember, help is only an e-mail away: cs187help@cs.umass.edu.

Learning Goals

- To install the Eclipse integrated development environment (IDE).
- To download starter code from Moodle.
- To import an Eclipse project into the Eclipse IDE.
- To run Java code in the Eclipse IDE.
- To run JUnit tests in the Eclipse IDE.
- To write Java code in the Eclipse IDE.
- To properly export a Java project from Eclipse.
- To properly upload and submit a Java Eclipse project into Moodle.

General Information

Reminder: Copying partial or whole solutions, obtained from other students or elsewhere, is academic dishonesty. Do not share your code with your classmates, and do not use your classmates' code.

You are responsible for submitting project assignments that compile and are configured correctly. If your project submission does not follow these policies exactly you may receive a grade of zero for this assignment.

Policies

- For some assignments, it will useful for you to write additional class files. Any class file you write that is used by your solution MUST be in the provided `src` directory you export.
- The TAs and instructors are here to help you figure out errors, but we won't do so for you after you submit your solution. When you submit your solution, **be sure to remove all compilation errors from your project**. Any compilation errors in your project will cause the autograder to fail, and you will receive a zero for your submission.

Test Files

In the `test` directory, we provide several JUnit test cases that will help you keep on track while completing the assignment. We recommend you run the tests often and use them to help create a checklist of things to do next. But you should be aware that we deliberately don't provide you the full test suite we use when grading.

We recommend that you think about possible cases and add new `@Test` cases to these files as part of your programming discipline. Simple tests to add will consider questions such as:

- Do your methods taking integers as arguments handle positives, negatives, and zeroes (when those values are valid as input)?
- Does your code handle unusual cases, such as empty or maximally-sized data structures?

More complex tests will be assignment-specific. To build good test cases, think about ways to exercise methods. Work out the correct result for a call of a method with a given set of parameters by hand, then add it as a test case. Note that we will not be looking at your test cases, they are just for your use.

Before submitting, make sure that your program compiles with and passes all of the original tests. If you have errors in these files, it means the structure of the files found in the `src` directory have been altered in a way that will cause your submission to lose some (or all) points.

Problem 1

Download the Starter Code

Download the provided archive file containing the starter code ("`hamspam-student.zip`") for this assignment and save it somewhere where you can find it. When you click on the link the zip file will be downloaded to your computer wherever downloaded files normally go. Move it into a useful place. You do not need to unzip it—Eclipse will handle that for you.

Install the Eclipse Development Environment

All the assignments in this course use the Eclipse Integrated Development Environment. The latest version is available at `http://eclipse.org/downloads/packages/eclipse-ide-java-developers/lunasr1a`. You can download the version you need by following that link. If you already have a version of Eclipse installed, please make sure you upgrade to the latest version.

Depending upon your operating system, you might need to install a Java Development Kit. You should be using the Standard Edition, version 7, sometimes abbreviated as the Java SE7 JDK. The latest version of this JDK is available at `http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`. You want the first thing that appears on that page, the Java SE Development Kit (probably version 7u71). Accept the license agreement and choose the appropriate version for your operating system.

After you have the correct versions of the JDK and Eclipse installed you can proceed to the next section.

Problem 1 continued on next page...

Update: We have had some reports from students of problems with Eclipse. These only occur if you download and install Java 7 after installing Eclipse. The symptom is that the downloaded project will show several items under the Problems tab (The project cannot be built until build path errors are resolved. and Unbound classpath container: JRE System Library [JavaSE-1.7] in project hamspam-student). Many items in the source will show a red underline, and nothing will build. The solution is to:

- **make sure that JDK 7 is installed,**
- **open Eclipse,**
- **to Eclipses preferences,**
- **expand the Java item on the left,**
- **select Installed JREs,**
- **click the Search button to add the Java7 JRE, and**
- **close the preferences window.**

Import the Code into Eclipse

Open the Eclipse Application

You will be asked to specify a location for your Eclipse workspace. This is a directory on your system where all of your Eclipse projects can be stored. You can use one workspace throughout the entire class, and this is where you will set it up. If you are opening Eclipse for the first time, you may see a welcome screen with links to instructions, tutorials, etc. You are welcome to read these, but eventually you should click on the upper right: there is a button that will take you to the "Workbench," the standard view for working in Eclipse.

Import the Starter Code

Choose "File → Import" from the menu. A window will come up so you can choose how to import. Select "General" and within that, "Existing Projects into Workspace." (It may seem strange, but do not choose "Archive File.") Then click "Next". Choose the button for "Select archive file" and locate the file you downloaded. above. Then click "Finish". You should see a `hamspam-student` Java project in the Package Explorer window on the left. Click on the triangles to reveal the content of this directory and the `src`, `support`, and `test` directories within it.

Make the `support` Directory Read-Only

The `support` folder contains support code that we encourage you to use (and which must be used to pass certain tests), but you are not allowed to change or add anything in this folder. To help ensure that you do not change anything there, we suggest that you set the `support` folder to be read-only. You can do this by right-clicking on it in the Package Explorer, choosing "Properties" from the menu, choosing "Resource" from the list on the left of the pop-up "Properties" window, unchecking the "Permissions" check-box for "Owner-Write", and clicking the "OK" button. A dialog box will show with the title "Confirm recursive changes", and you should click on the "Yes" button.

You should see three Java source files among the directories:

File	Location	Description
`HamSpam.java`	`src`	The behavior and state associated with ham and spam integers and arrays.
`HamCommander.java`	`support`	A command-line program to drive `HamSpam`.
`HamSpamTest.java`	`test`	An incomplete set of tests for `HamSpam`.

Problem 1 continued on next page...

You can open them by double-clicking on them. Do so, and take a look through each of them. You'll quickly note they involve a game about "Ham", "Spam", and numbers.

On Ham and Spam

"Ham and Spam" is a children's counting game. Before it is played, the players agree on a *ham number* and a *spam number*. Both are integers greater than one, and they cannot be the same number. The players then take turns saying the *hamspam* value for each successive integer, starting at one. But:

- If the number they are supposed to say is divisible by the ham number and not the spam number they say "ham" instead of the number.
- If the number they are supposed to say is divisible by the spam number and not the ham number they say "spam" instead of the number.
- If the number they are supposed to say is divisible by both the spam number and the ham number they say "hamspam" instead of the number.

For example, if the ham number is three, and the spam number is four, then the first twelve hamspam values are:

1, 2, ham, spam, 5, ham, 7, spam, ham, 10, 11, hamspam

For this assignment, you are going to modify the `HamSpam` class provided to produce the correct value or values for a game of "Ham and Spam".

Using Eclipse

Running the code.
First explore the code as it is. Under `support`, choose the `HamCommander.java` file, then click the green play button in the top row of the Eclipse window, or choose "Run" from the Run menu. A console will appear in the bottom of the Eclipse window. Follow along, pressing Enter after each number you enter, to see the (sometimes incorrect) results of the current implementation of Ham and Spam.

Testing the code.
Next, choose `HamSpamTest.java` in the Package Explorer and run it using the play button or the menu, as above. The package explorer on the left will switch to a JUnit pane, which will show the testing output. You should see a total of seven tests: three tests that pass, and four tests that fail. Familiarize yourself with the testing interface. If you select the first failed test, you should see the following under Failure Trace:

```
org.junit.ComparisonFailure: getValue returns incorrect value
expected:<[hamspam]> but was:<[12]>
at hamspam.HamSpamTest.testHamAndSpamGetValue(HamSpamTest.java:41)
```

(You may have to resize the JUnit pane to see the full message.)

Diagnosing and Fixing the Problems

JUnit tests work by checking that the expected result of a method call equals the actual result. The failed test indicates that one (or more) methods in the starter code aren't returning correct values. Examine the test; you can double-click on `at hamspam.HamSpamTest.testHamAndSpamGetValue(HamSpamTest.java:41)` to jump to the test in the source pane. Here, you can see that the test case expects the result of `hamspamThreeFour.getValue(12)` to be the `String "hamspam"`, but as the output in the Failure Trace indicates, it produced the `String "12"`.

Go to the declaration of `getValue()`; you can do so by right-clicking on it and selecting "Open Declaration", or by double-clicking on `HamSpam.java` in the Package Explorer and scrolling to it.

Your goal should be to correct the implementation of the `getValue()` method. The correct solution is not merely to make the function return `"hamspam"` when `n == 12`. Instead, you should revise the code so that the function will return the correct `String` for any `n`.

There are at least two things to correct. First, the `HamSpam` class constructor takes a ham number and a spam number, but it doesn't store them as state (for example, as private instance variables). Second, the `getValue()` method checks currently checks for equality against numbers, not divisibility. You can use the modulus (remainder) operator (`%`) to check for divisibility. It divides one number by another and returns the remainder. For example, if you wanted to print out whether the value of a variable n was divisible by three, you could write:

```
if ((n % 3) == 0) {
    System.out.println(n + "␣is␣divisible␣by␣3");
else {
    System.out.println(n + "␣is␣not␣divisible␣by␣3");
}
```

When we grade your program, we will test both `getValue()` and `getValues()` with other ham and spam numbers to make sure each works properly. Our test cases will obey the constraints described in this assignment (for example, the ham number and spam number will always be greater than one, and never be equal to one another), but are otherwise unconstrained.

We suggest you add a few additional test methods (remember the `@Test` decorator) to check your work. You can also use the `HamCommander` class for interactive testing.

Export and Submit

When you have completed the changes to your code, you should export an archive file containing the entire Java project. To do this, click on the `hamspam-student` project in the package explorer. Then choose "File → Export" from the menu. In the window that appears, under "General" choose "Archive File". Then choose "Next" and enter a destination for the output file. Be sure that the project is named **`hamspam-student`**. Save the exported file with the `zip` extension (any name is fine). Log into Moodle and submit the exported zip file.