

Lambda Hands-on Lab

Stuart Marks   
Principal Member of Technical Staff, Oracle

Simon Ritter   
Technology Evangelist, Oracle

Angela Caicedo   
Java Evangelist, Oracle

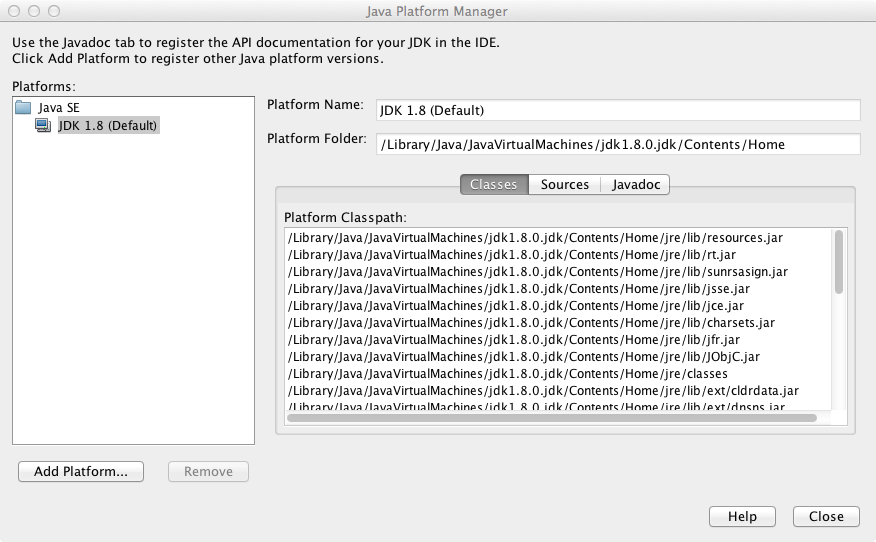
**1. Getting Ready**

1.1. **Install NetBeans 7.4** (https://netbeans.org)

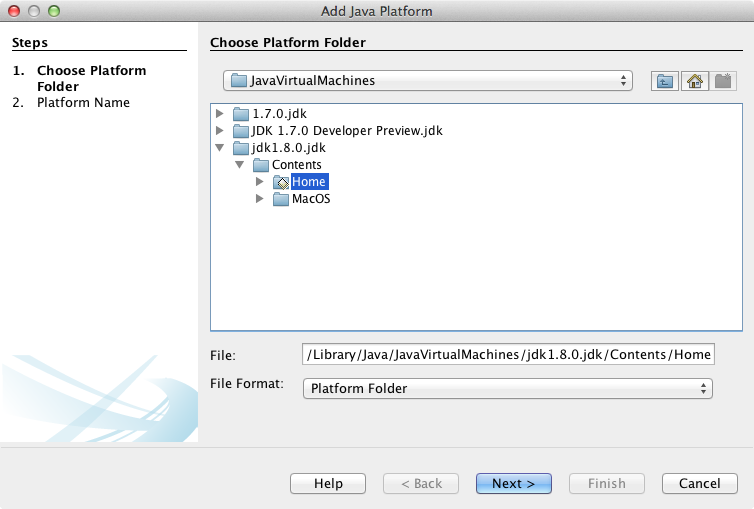
* Make sure installation is successful.

1.2. **Install JDK 8 Early Access latest release.** (https://jdk8.java.net/download.html)

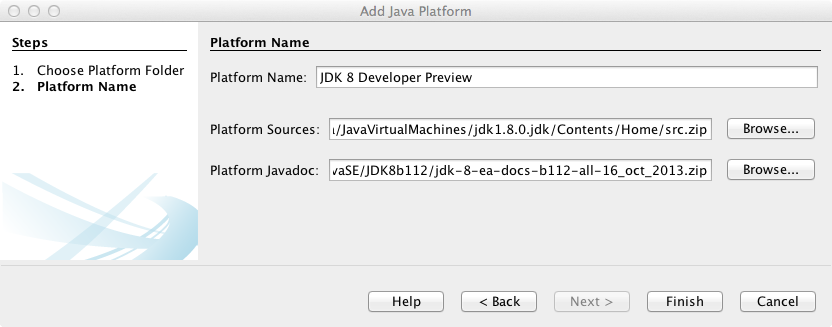
* By the time this documentation was done, the latest build was b115.
* Make sure you also download the JDK zip documentation.

1.3. **Configure NetBeans** to use the installed JDK.

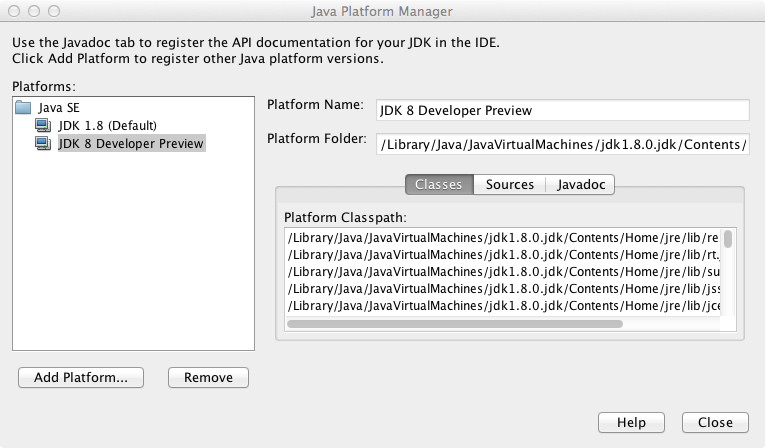
* From the top menu select **Tools** and then select **Java Platforms**
* Click on Add Platform
* Navigate and select the installation directory for the JDK 8, and click on Next.



* Enter **JDK 8 Developer Preview** for Platform Name.
* Specify the location for the Platform Javadoc: browse and select the location of the JDK documentation you downloaded.



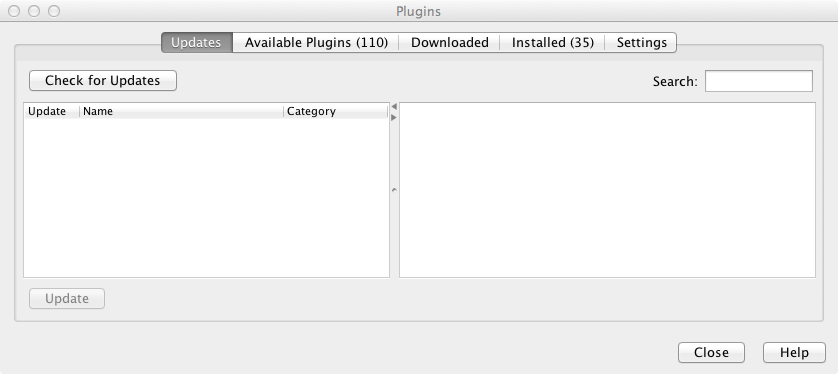
* Click on Finish.
* You should see now a new listed Java SE platform



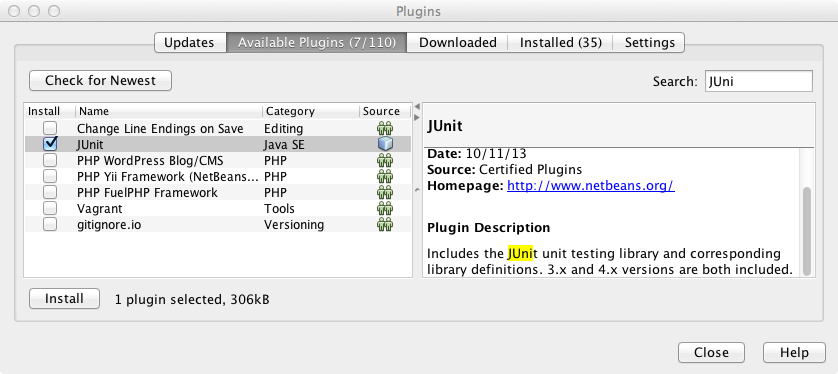
* Close your Java Platform Manager.

**1.4. Installing JUnit with NetBeans Update Center**

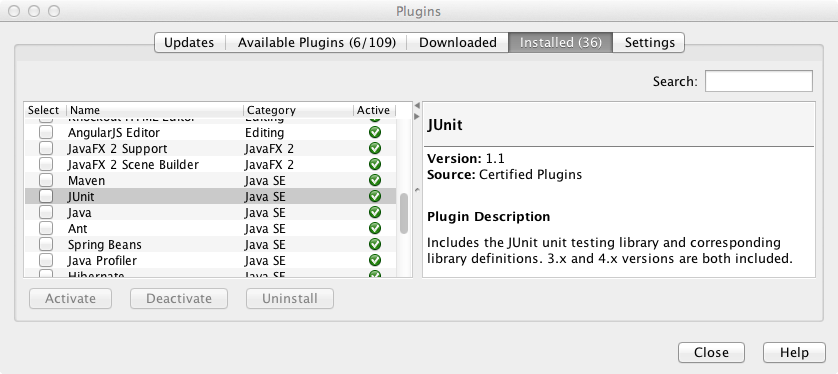
* From the main menu, select **Tools > Plugins**
* Click on **Available Plugins** at the Plugins window



* Type **“JUni”** in the Search Text area. (Don’t worry if the list from the image looks different from your list)
* Click the checkbox for JUnit installation
* Click Install



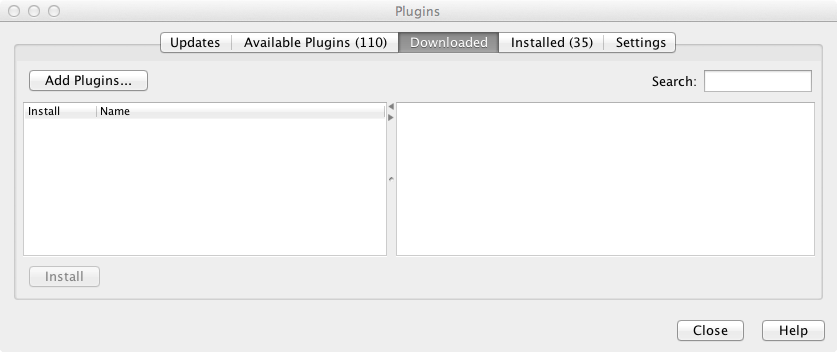
* The NetBeans IDE installer gets displayed. Just follow the instructions for the installation.
* Once the installation finished, make sure JUnit is now part of the Installed plugins and it’s **Active**

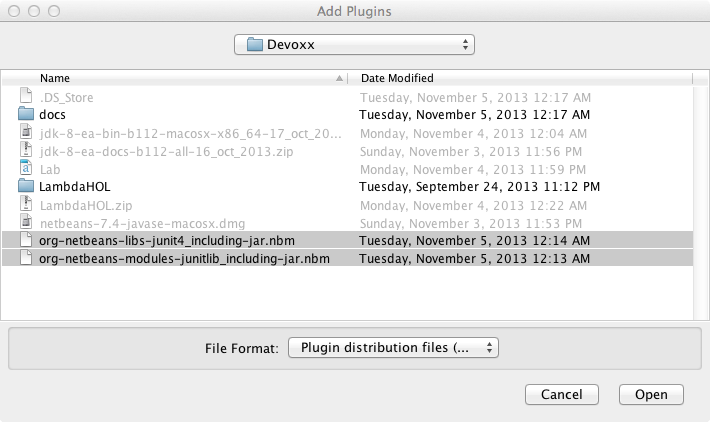


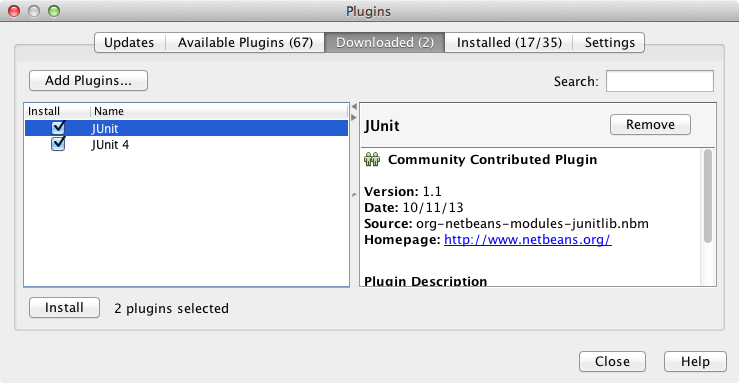
* Close the plugin window.

**1.5. Installing JUnit offline**

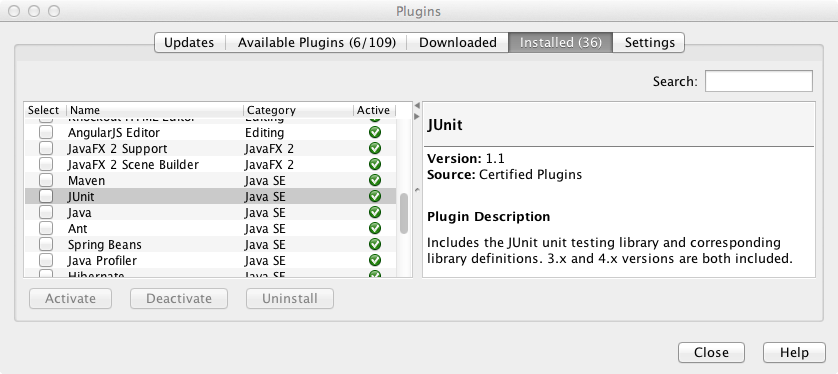
* If you don’t have Internet connection, you can still install the JUnit plugin from the provided software for this lab.
* In the lab software, you should have two files named: **org-netbeans-libs-junit4\_including-jar.nbm** and **org-netbeans-modules-junitlib\_including-jar.nbm**.
* From the main menu, select **Tools > Plugins**
* Click on **Downloaded** at the Plugins window
* Click on **Add Plugins…** button



* Look for the lab material and select **org-netbeans-libs-junit4\_including-jar.nbm** and **org-netbeans-modules-junitlib\_including-jar.nbm** files.
* Click on Open
* Now you should see JUnit and JUnit 4 selected for installation
* Click on Install button



* The NetBeans IDE installer gets displayed. Just follow the instructions for the installation.
* Once the installation finished, make sure JUnit is now part of the Installed plugins and it’s **Active**

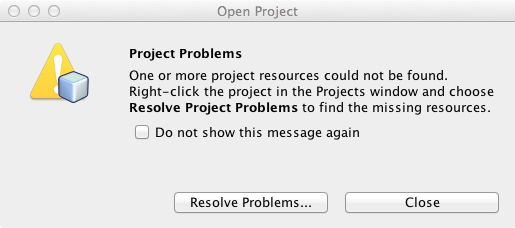


* Close the plugin window.

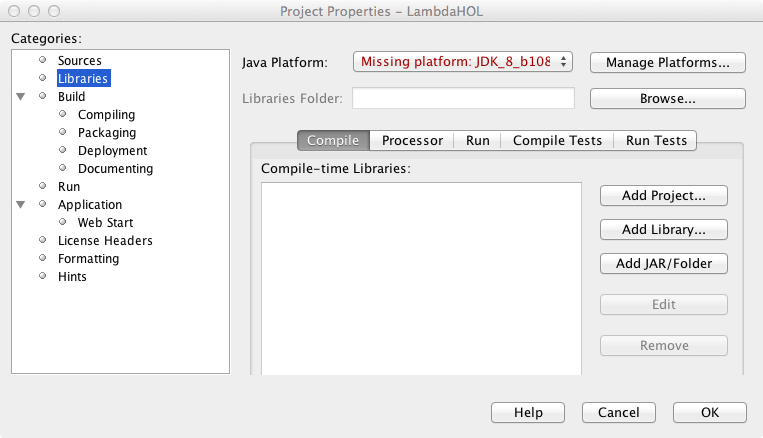
**2. Starting with the exercises**

**2.1 Open LambdaHOL project**

* You might get a message about missing some project resources. If this is your case, follow the next steps to fix the project, otherwise continue with the exercises.
* Close the warning window



* Right click on your project, and select **Properties**
* From the **Categories** list, select **Libraries**
* On the right hand side, you might notice that the Java Platform is missing. This error is cause if you used a different name for the new added Java Platform, or because some misspelling in the Platform name.



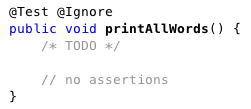
* From the Java Platform drop-down list select the **JDK 8 Developer Preview**, and click Ok.
* Now your project should show no errors.

**2.2. Exercise Overview**

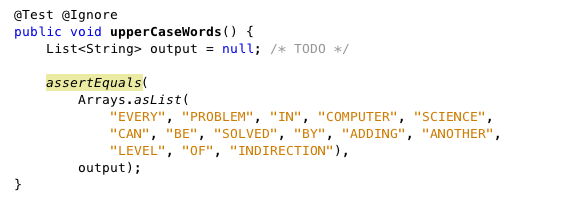
The project is structured as a set of JUnit tests. Open the Test Packages folder, and in the "exercises" package, open the file Exercises.java.   
  
Each exercise (except for the first) is a single test with a problem statement in a comment, a local variable with a TODO comment next to it, and some assertions at the end. Each test exercise has an @Ignore annotation on it, which causes the JUnit framework not to run the test. Run the tests in this file using Alt-F6 (Cmd-F6 on Macintosh). If you do this now, the result should be no tests passed, no failures, and 12 tests skipped.   
  
Your task is, for each exercise, to replace the TODO comment with code so that the assertions are satisfied. Remove the @Ignore annotation from a test in order to have JUnit run it.   
  
The first exercise doesn't have any assertions. It's just there to help you get started. It's also a useful place to try out some experiments without risking getting a red bar.   
  
Continue through the exercises, removing each @Ignore tag and replacing the TODO comment with code to make the test work. Try to keep the bar green! When you're finished, all 12 tests should pass, and none should be skipped.   
  
There are two additional files in the exercises.answers package. The first, ExerciseAnswers.java, is the set of tests with an answer filled in. Look in here if you want some hints as to how to proceed, or if you want to check your answer against what the lab's authors came up with. There are often variations of how the APIs can be put together, so don't worry if you don't come up with exactly the same solutions we did.   
  
The other file in exercises.answers is ExercisesOrig.java. This is a copy of the original state of the Exercises.java file. You can copy stuff from here if you've accidentally messed up your copy of Exercises.java and you want to restore it.

**2.3. API Hints**   
  
In the API specification (javadoc) most of new APIs are in the java.util.stream package. The Stream interface itself has most of the interesting methods.   
  
To get a stream from a collection, use Collection.stream() or Collection.parallelStream(). Other useful ways to get streams are IntStream.range(), Stream.of(), Arrays.stream(), BufferedReader.lines(), and CharSequence.chars(), and Pattern.splitAsStream().

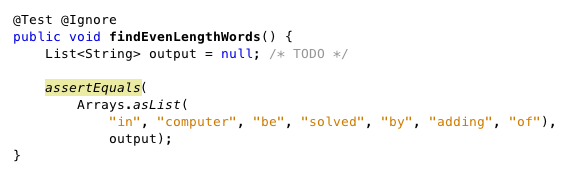
**2.4. Exercises**

**Exercise 1:** Print out all the words in wordList, which is a static List<String> defined at the bottom of this file.

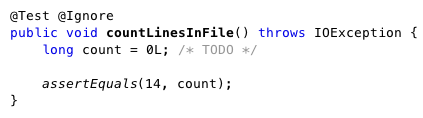
**Exercise 2:** Convert all words in wordList to upper case, and gather the result into an output list.



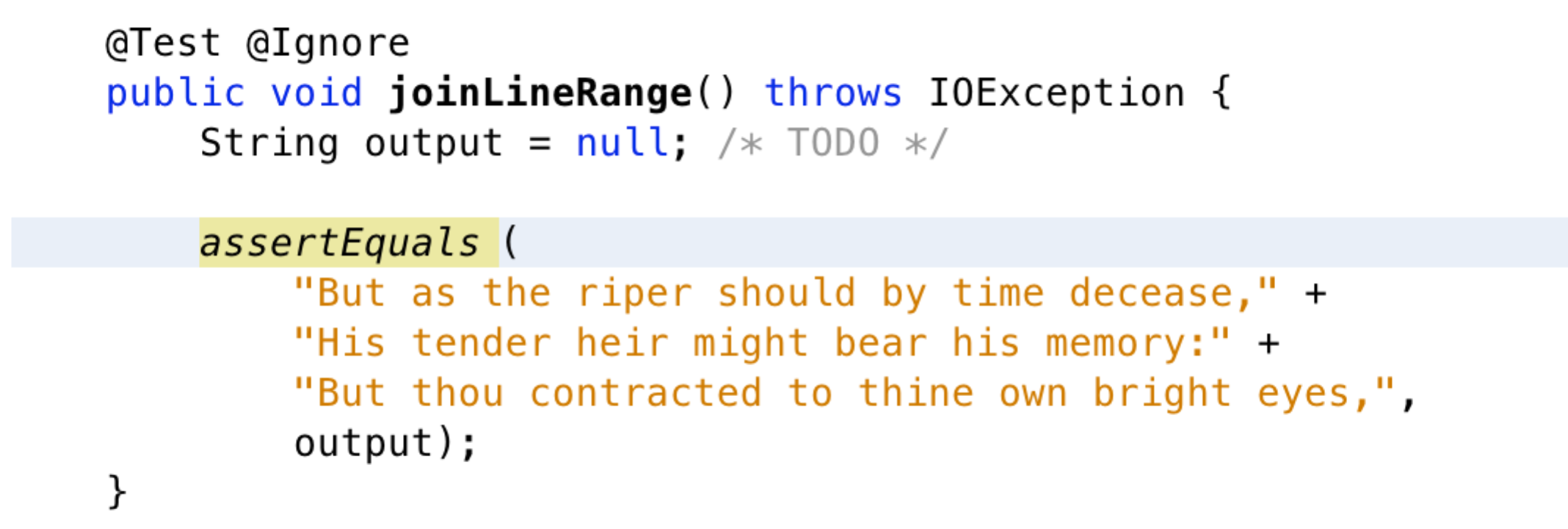
**Exercise 3:** Find all the words in wordList that have even length and put them into an output list.



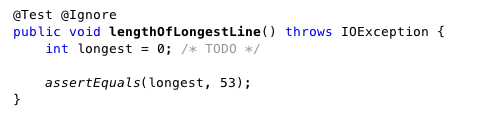
**Exercise 4:** Count the number of lines in a file. The field \*reader\* is a BufferedReader which will be opened for you on the text file. See the JUnit @Before and @After methods at the bottom of this file. The text file is "SonnetI.txt" (Shakespeare's first sonnet) which is located at the root of this NetBeans project.



**Exercise 5:** Join lines 3-5 from the text file into a single string.



**Exercise 6:** Find the length of the longest line in the file.



**Exercise 7:** Collect all the words from the text file into a list.

**Hint:** use String.split(REGEXP) to split a string into words. Splitting this way results in "words" that are the empty string, which should be discarded. REGEXP is defined at the bottom of this file.

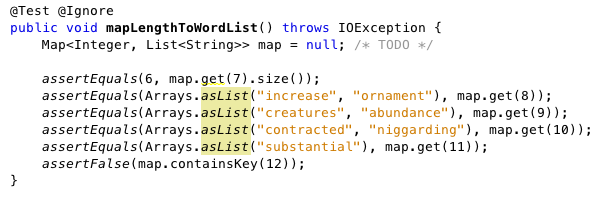


**Exercise 8:** Create a list containing the words, lowercased, in alphabetical order.

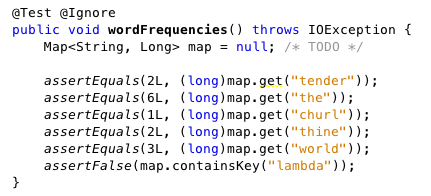


**Exercise 9:** Sort unique, lower-cased words by length, then alphabetically within length, and place the result into an output list.

**Exercise 10:** Categorize the words into a map, where the map's key is the length of each word, and the value corresponding to a key is a list of words of that length. Don't bother with uniqueness or lower-casing the words.



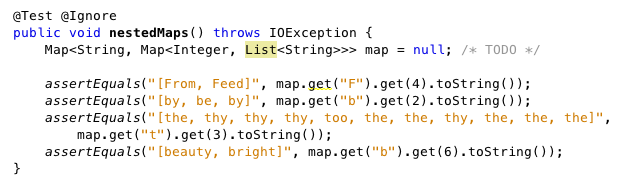
**Exercise 11:** Gather the words into a map, accumulating a count of the number of occurrences of each word. Don't worry about upper case and lower case. Extra challenge: implement two solutions, one that uses groupingBy() and the other that uses toMap().



**Exercise 12:** Create nested maps, where the outer map is a map from the first letter of the word to an inner map. (Use a string of length one as the key.) The inner map, in turn, is a mapping from the length of the word to a list of words with that length. Don't bother with any lowercasing or uniquifying of the words.

For example, given the words "foo bar baz bazz" the string representation of the result would be:

{f={3=[foo]}, b={3=[bar, baz], 4=[bazz]}}.



**Test Infrastructure**



**3. Web Links**   
  
Java Tutorial on Lambda Expressions:   
    <http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>   
  
Lambda FAQ:   
    <http://www.lambdafaq.org/>   
  
State of the Lambda (language design background and rationale):   
    [http://cr.openjdk.java.net/~briangoetz/lambda/lambda-state-final.html](http://cr.openjdk.java.net/%7Ebriangoetz/lambda/lambda-state-final.html)   
  
State of the Lambda Libraries (library design):   
    [http://cr.openjdk.java.net/~briangoetz/lambda/lambda-libraries-final.html](http://cr.openjdk.java.net/%7Ebriangoetz/lambda/lambda-libraries-final.html)   
  
Oracle Learning Library webcast (Michael Williams):   
    <http://www.youtube.com/watch?v=giyKPnrpcjA>   
  
Jump-Starting Lambda Programming presentation (Stuart Marks, JavaOne 2012):   
    <http://www.youtube.com/watch?v=bzO5GSujdqI>   
  
Jump-Starting Lambda Programming slides only (Stuart Marks, JavaOne 2013):   
<http://stuartmarks.files.wordpress.com/2013/09/tut3877_marks-jumpstartinglambda-v6.pdf>