

**CS 310 Data Structures**  
**Programming Assignment 1**  
**Fall 2009**

Due Wednesday, Oct. 7 at midnight

**Purpose**

This assignment aims to help you:

- Learn about using Maps
- Review string tokenization and manipulation
- Review packages
- Start thinking about performance and memory use

**Reading**

Read Weiss Chap. 6 to pg. 239, Chap 12, Sec. 12.2 (On Xref.java)

**Description**

1. Build Xref by downloading its sources from Weiss's website, linked to the class homepage where the text is specified. Replace "weiss" in the imports to "java", and compile the sources. Run Xref on Xref.java and check its results by "grep -n lineItr Xref.java" on UNIX or "find /n "lineItr" Xref.java" on Windows (in a command window), and try a few other grep/find's. Note that you can search Xref's output by grep/find: try "java Xref Xref.java | grep lineItr" or "java Xref Xref.java | find -n "lineItr".

2. Using Packages. See Weiss, Sec 3.6 and the [Sun tutorial on packages](#).

Add "package cs310" to the top of the two .java files and set up a proper project with directory structure as follows:

In directory pa1, subdirs classes and src, each with cs310 subdirs for the package files. Here is output from the Window "tree" command:

```
|—classes
|  |—cs310
|—src
|  |—cs310
```

Now you can compile everything by

```
cd src (the top-level source directory)
javac -d ../classes cs310/*.java
cd ../classes (the top-level classes directory)
java cs310.Xref
```

The reason to be cd'd to these directories is that this is the easiest way to get the right classpath in use. The default classpath is the current directory, and you've cd'd to the right place to set the desired classpath this way. You can do the commands from other directories by specifying the classpath on the command line, like this:

```
javac -cp src cs310/*.java (while cd'd to pa1)
```

For DrJava or eclipse, set up the directories as above, put the sources in src/cs310, add the package statements to \*.java, and then set up a project in the IDE. For DrJava, see [packages in DrJava](#). Check the “recursive” checkbox in Open Folder window.

In memo.txt, show output of command line compile and run in this project, and tree output (or du for UNIX/MacOSX). Describe the display of the project source filenames in your IDE.

3. Having studied Xref.java, now figure out how to recode the qualifying exam problem using a Map for each terminal line, on the assumption that the size of the problem is dominated by the number of input lines, not the 500 terminal lines. Use a package named cs310 as in problem 2. Consider whether to use a HashMap or TreeMap, and what the key should be. Usage (as before except for package use): java cs310.TermReport<data.txt.

4. Problem 12.17, pg. 449, the SpellChecker. Use the dictionary /usr/dict/words from UNIX, available via your cs310 UNIX login, and make up a personal dictionary. Again use package cs310 (so all the pal source files are in the same directory.) Note that a personal dictionary just adds to the set of correctly-spelled words. For parsing the words from the input file, don't try to use Tokenizer.java, but instead use java.util.Scanner somewhat like the qual solution. Try to write efficient string manipulation code for the task of checking modified strings when the user-provided string does not match in either dictionary. Don't add Strings together character by character, but rather use a StringBuffer. Read the JDK documentation to see how a StringBuffer is converted to a String. Assume the size of the problem is dominated by the big dictionary. NOTE: Usage: java cs310.SpellChecker input-doc main-dictionary personal-dictionary. Also note that if you work with another UNIX machine the dictionary may be installed somewhere else. Look for it using: find . -name words -print from the root directory.

### memo.txt

In the file memo.txt, answer (in short!) the following questions, in one to three pages (60-180 lines) of text. Use complete sentences, as you would for an English class

1. Discuss your experiences in writing these programs. What development tools (IDE, etc.) did you use? Did you develop on UNIX, or if on your own PC, did you have any problem recompiling and running on UNIX?
2. As described in problem 2 above.
3. Try to determine from your code reading whether you can switch from TreeMap to HashMap in GenerateCrossReference() (pg. 444). Write down your analysis. Try out the switch to see if your analysis is correct.
4. Describe your analysis of the qualifier problem and choice of Map there.
5. Compare the performance of the qualifier problem implemented with lists vs. maps, for N input lines.
6. Estimate the maximum memory in use for 500 lines and N input lines, assuming 90% of observations show the user on their assigned terminal, and 10% are random.
7. Analyze the CPU performance of the spell-checker, for N words in the big dictionary and O(1) words to check, and O(1) words in the personal dictionary. Also estimate memory use.

### Delivery

Before the due date, assemble files in the pal subdirectory of your provided cs310 directory on our

UNIX site. Make sure they compile and run on UNIX! They should, since Java is very portable. It's mainly a test that the file transfer worked OK.

- pa1/memo.txt (plain txt file, try “more memo.txt” on UNIX)
- pa1/classes/cs310/\*.class (compiled sources)
- pa1/src/cs310/Xref.java
- pa1/src/cs310/Tokenizer.java
- pa1/src/cs310/TermReport.java (as provided, or edited)
- pa1/src/cs310/LineUsageData.java (changed, for grading)
- pa1/src/cs310/Usage.java (as provided)
- pa1/src/cs310/SpellChecker.java (for grading)
- pa1/src/cs310/\*.java (any other sources for SpellChecker)

Check your filenames: login on UNIX: cd cs310/pa1, then “ls” should show memo.txt, src, classes. “ls src/cs310” should show all the source filenames. Remember that case counts on UNIX!