Laboratory 12

CS 1323, Spring 2015

# Learning Objectives

1. Declare and construct an array of String objects where the size of the active portion of the array may change during execution (oversize array). (10 points)
2. Insert elements into an oversized array of objects. (10 points)
3. Perform spelling checking using a global dictionary and a personal dictionary, updating the personal dictionary with user selected words. (20 points)
4. Save an oversized array to a file. (10 points)
5. Modify existing code written by someone else. (10 points)
6. Create at least two additional meaningful methods that pass arrays as parameters and/or return them. (20 points)
7. Write a program beginning with provided code written by someone else. (10 points)

10 points will be awarded for the documentation of your program. That means using good names for variables, comments, proper and consistent indentation of code, and meaningful use of whitespace.

Section 10: When your program is completed and running, have the teaching assistants check it to get credit for the lab. If you do not complete the laboratory during the allotted time, you may submit it on Janux before Monday, April 13th at 11:59 p.m. Only people who attend the whole laboratory will be permitted to submit assignments on Janux.

Section 1: Submit your finished project on Janux by Monday, April 13th at 11:59 p.m.

# Description

Many good word processing programs have two separate dictionaries for spelling checking: the global dictionary and a personal dictionary. The personal dictionary contains words that are correctly spelled that are not in common usage. For example, I use technical jargon in a lot of my writing. Most word processors don’t know that “accessor” and “mutator” are spelled correctly. But they are spelled correctly and I know they are spelled correctly. It would be nice for them not to be flagged as wrong every \*&^%$#@ time.

To work around this problem, a word processor can keep a personal dictionary. The personal dictionary is used in the same way that that the global dictionary is used—except that words can be added to it. One of my favorite old word processor programs gave you the option of adding any word that it thought was spelled wrong that you subsequently confirmed to your personal dictionary. After using that program for a number of years, spelling checking was really great.

Here is a sample interaction.

Enter a word or QUIT to stop:

hello

That word is spelled correctly

Enter a word or QUIT to stop:

goodbye

That word is spelled correctly

Enter a word or QUIT to stop:

jambone

That word is not spelled correctly

Would you like to add it to your personal dictionary Yes/No

yes

Enter a word or QUIT to stop:

jambone

That word is spelled correctly

Enter a word or QUIT to stop:

quit

The array that stores the personal dictionary will need to be allocated to a large enough size (100 is fine for this project). The active size will need to be stored and manipulated. You do not need to consider what will happen if more than 100 personal dictionary words are entered, although an error message to the user is always considerate.

Remember to create an empty PersonalDictionary.txt file for your program so that it will find the file the first time it runs.

# Writing A File

Writing files is similar to reading them. The class that writes simple files is called PrintWriter. It is constructed as below:

PrintWriter pw = new PrintWriter(new File(filename));

Any method that constructs a PrintWriter or calls a method that constructs a PrintWriter will have to throw a FileNotFoundException.

To write a String to a PrintWriter object use the code below:

pw.write(“Any String\n”);

It is critically important that you remember to close files when you are writing them. If you do not close a file, the file contents will be lost.

pw.close();

# Command Line Arguments (Optional)

This project allows you an opportunity to final use String[] args for something useful! The parameter to the main program is called a command line argument, although the rationale for this name only makes sense if you program in Java without an IDE like eclipse. If you have a full installation of Java, you can run a Java program from the Command Line. First, export your Java file from eclipse and store it on your desktop. To compile this .java file into a class file, open the command prompt, navigate to the desktop (cd directoryName will change directories), and compile your code by entering:

javac MyClass.java

If you look at your desktop, you will now see a file called MyClass.class has been added. This is the compiled version of your Java program in a language called bytecode. You run this program by entering:

java MyClass

When you run a program this way, you have the option of entering (at the command line) additional arguments—hence command line arguments. For example, you could enter:

java SpellCheck globalDictionary.txt personalDictionary.txt

Inside your program, the String array args will now have a length of 2.

args[0] will be “globalDictionary.txt”, the name of the global dictionary

args[1] will be “personalDictionary.txt”. You can use these Strings to construct your Scanner objects that read files.

Doing this in eclipse is more complicated. First go to Project -> Properties . Select the Run/Debug Setting category, and create a New Setting. Eclipse will ask you if you want a Java Applet or a Java Application. We want a Java Application, so select that and click OK.

You will see another tabbed interface appear. Select the Arguments tab and enter the file names. These will be fed in as command line arguments when your program runs.

The purpose of command line arguments is to provide another communication channel to your program. Things like resource file names are nicely handled as command line arguments.