Assignment for Week 11: Maps Et Cetera

Due Wednesday, 7 April 2004

Goals

This assignment consists of a small program that uses several and possibly many interfaces and classes from Java's standard class libraries. The goal is to give you a bit of design freedom: You get to decide which parts of the standard libraries you want to use. If you design well, then this assignment should be fairly straightforward. If you design less well, then this assignment might become more interesting.

Hints

Some or all of the following interfaces and classes are likely to prove useful. In the `java.lang` package: `Comparable`, `Integer`. In the `java.util` package: `Collection`, `Comparator`, `Iterator`, `List`, `Map`, `Set`, `Collections`.

The Application

Have you ever wondered about the size of Shakespeare's vocabulary? For this assignment you will write a program that reads its input from a text file and lists the words that occur most frequently, together with a count of how many different words occur in the file. If this program were run on a file that contains all of Shakespeare's works, it would tell you the approximate size of his vocabulary, and how often he uses the most common words.

Hamlet, for example, contains about 4542 distinct words, and the word "king" occurs 202 times.

The Assignment

You are given a text file `test.txt` that contains *Hamlet*, and a Java file `Week11.java`, which is in the default package. Your assignment is to write another Java file `Word.java`, which is also in the default package.

1. Define a class `StudentTests`. Within this class, define a `run()` method that will run all of the tests you write for this assignment.

2. Define a class `Word` that represents an English word. The constructor for `Word` shall take one argument, a string such as "king".

3. The `Word` class shall implement the `Comparable` interface, and shall override the `toString()`, `equals(Object)`, and `hashCode()` methods

of the `Object` class. Two instances of `Word` shall be considered equal if the strings that were passed to their constructors are equal. Similarly a `Word`'s `compareTo` method should just delegate to the corresponding `String`'s method.

4. Define a class `WordCounter`. The constructor for `WordCounter` shall take no arguments, and shall initialize the `WordCounter` object in whatever way you deem appropriate for a `WordCounter` object that has not yet counted any words.

5. Within the `WordCounter` class, define the following methods:

```
// Records the Word objects generated by the given Iterator.
void countWords (Iterator it) { ... }

// How many different Words has this WordCounter recorded?
int words () { ... }

// Prints the n most common Words and their frequencies.
void printWords (int n) { ... }
```

6. The `countWords(Iterator)` method takes an `Iterator` that generates `Word` objects, and updates the state of this `WordCounter` to remember all of the generated `Word` objects together with how many times each `Word` was generated. The order in which the `Word`s were generated does not matter, and does not need to be remembered by this `WordCounter`.

7. The `words()` method returns the number of *different* `Word`s that have been generated by the `Iterator`s that have been passed thus far to this `WordCounter`'s `countWords` method. This is usually less than the total number of `Word`s that have been generated. For example, the word "and" is only one word, even if it has been generated 966 times.

8. The `printWords` method uses `System.out.println` to print the n most common `Word`s that have been generated by the `Iterator`s that have been passed to this `WordCounter`'s `countWords` method, together with the number of times each `Word` has been generated. Each `Word` should be printed on the same line as its frequency, separated by a single space, one `Word` per line, in the format shown by this example output:

```
chuck 4
wood 4
a 2
could 1
how 1
if 1
much 1
would 1
```

The `Words` shall be printed in order of decreasing frequency, as shown above. When two or more `Words` occur with the same frequency, such as "chuck" and "wood" above, the `Words` should appear in alphabetical order. If there are fewer than n words altogether, then the output shall end after all of the `Words` and frequencies have been printed.