SPRING 2003: **COP 3530** DATA STRUCTURES
[PROGRAMMING ASSIGNMENT 5; DUE APRIL 17 IN CLASS.]

Problem Description

Your task is to write a program to read Shakespeare's play *Hamlet* (`http://www.cs.fiu.edu/~giri/teach/3530/s03/Prog5/HamletComplete.txt`) and create a hash table of all the words in the text. After the text is processed, your program should print out (1) the number of lines processed, (2) the final size of the hash table, (3) the number of entries (words) stored in the hash table, (4) the load factor of the hash table, (5) the size of the largest cluster, (6) the average size of a cluster, (7) the number of clusters, and (8) the most frequent word that appeared in the text.

In the second part of the program, your program should read a set of query words from the data file `Query.txt`. Search in the hash table for each of the words mentioned in that datafile. For each word, report (1) whether the word was found or not, (2) the number of entries of the hashtable that were accessed for each search, (3) the word itself, (4) the number of times it appeared in the text, and (5) its hash value. The output should be neatly formatted.

Input

Your program cannot use a local copy of the data file containing the Hamlet text. Instead it must use the data file directly from the world wide web from the URL:
`http://www.cs.fiu.edu/~giri/teach/3530/s03/Prog5/HamletComplete.txt`

All words in upper case letters are not to be considered as part of the text. For instance names of the speakers such as `BERNARDO` or `HORATIO` are not to be put into the hash table. Words should be stripped of punctuations. For instance the tokenizer might give you a token such as "`castle.`" or "`king!`" or "`longer--married`". The punctuations should be removed before inserting the word into the hash table, leaving the words as "`castle`" or "`king`". The string "`longer--married`" should generate two words "longer" and "married". Strings with hyphens or apostrophes are considered as one word (for e.g., "`to-night`", "`'tis`", and "`we'll`" are single words).

The query file is called `Query.txt` and can be downloaded from the class website as usual. You are free to use a local copy of this file.

Strategy

You must use the class `HashSet` available from the course website. This is a modification of the class `HashSet` from your text, which implements a hash table. `HashSet` uses *quadratic probing* for collision detection (as given in the text). You are also required to fill in two methods in the `HashSet` class. You may not change anything else in the `HashSet` class. Study this class carefully, and make sure you understand all the modifications made to it (all the changes are commented). In particular, a new constructor has been added to it. This constructor allows you to construct a `HashSet` object with a specified table size and specified limit for the load factor.

You must use a `Word` object. This object should contain the word (String type) you are storing. It will also contain a field for the frequency of occurrence of the word. Also, this class **must** implement the methods `toString, equals`, and `hashCode` (these are required, since otherwise the `HashSet` method will not function properly). Two words are considered equal if their strings are the same with the case ignored. Thus, for example the words “`'Tis`” and “`'tis`” are to be treated as the same word. For implementing the `hashCode` method, simply invoke the `hashCode` method for the `String` object from Java API.

Note that the `HashSet` data structure you will use will contain entries of type `Word`.

To read the Hamlet file using Java's `URL` class, the following statements can be used.:

```
String urls = "http://www.cs.fiu.edu/~giri/teach/3530/s03/Prog5/HamletComplete.txt";
URL u = new URL(urls);
URLConnection connection = u.openConnection();
HttpURLConnection httpConnection = (HttpURLConnection) connection;

// check if response code is HTTP_OK (200)
int code = httpConnection.getResponseCode();
if (code != HttpURLConnection.HTTP_OK)
{
   String message = httpConnection.getResponseMessage();
   System.out.println(code + " " + message);    // read server response
   return;
}

InputStream in = httpConnection.getInputStream();
BufferedReader fIn = new BufferedReader(new InputStreamReader(in));
```

What to Submit

Run the program three times on the same text (`HamletComplete.txt`) and the same query file (`Query.txt`), but with different parameters for the hash table. The three runs should use:

1. Table Size = 5167, LoadFactorLimit = 1.0
2. Table Size = 6823, LoadFactorLimit = 1.0
3. Table Size = 6823, LoadFactorLimit = 0.7

As usual, also submit the source code for your program and the output of `Javadoc`. Make sure you also include a print out of your modified `HashSet` class. Your floppy diskette should contain all the requisite `.java, .class, .html, .dat, .out` files that are relevant for the grader to check the program. Do not include the large Hamlet text on the diskette. Make sure that the hard copy you submit is the same as the copy on the floppy, and is compilable from the floppy. Also include a **signed** statement swearing that the submitted work is your own.

Output

As mentioned above, the output should be neatly formatted. A sample output format can be found on the course web page (see `SampleOoutput.txt`). Make sure you debug the program with a simple and small sample text file. Make sure the entries are inserted correctly in the table. To help you with this, there is flag called `DEBUG` in the `HashSet` class. If this flag is set using the `setDEBUG` method, then the `HashSet` method will give you more detailed output. In the final output you submit, make sure these debug messages are NOT PRINTED OUT!

Challenges for the bored

For extra credit, you could try the following problems:

Easy When searching for a word, also output the line numbers where it appears.

Easy Output the ten most frequent words that are not prepositions or conjunctions or names.

Medium Draw pictures of occupied cells of the type shown in Figure 20.5 (page 690) in your text.

Hard Animate the picture of occupied cells as the program reads through the text.

Hard Output the ten pairs of words that appear most frequently on the same line.

Hard Output the ten most frequent 3-word phrases in the text.