Lab 9: Sets in the Java Collection Framework

For this week's lab, you will use two of the classes in the Java Collection Framework: *HashSet* and *TreeSet*. You will use these classes to implement a spell checker.

Set Methods

For this lab, you will need to use some of the methods that are defined in the Set interface. Recall that if set is a Set, then the following methods are defined:

- **set.size()** -- Returns the number of items in the set.
- **set.add(item)** -- Adds the item to the set, if it is not already there.
- **set.contains(item)** -- Check whether the set contains the item.
- **set.isEmpty()** -- Check whether the set is empty.

You will also need to be able to traverse a set, using either an iterator or a for-each loop.

Reading a Dictionary

The file words.txt (in the code directory) contains a list of English words, with one word on each line. You will look up words in this list to check whether they are correctly spelled. To make the list easy to use, you can store the words in a set. Since there is no need to have the words stored in order, you can use a *HashSet* for maximum efficiency.

Use a Scanner to read the file. You can create a scanner, *filein*, for reading from a file with a statement such as:

```
            filein = new Scanner
(new File("/classes/s09/cs225/words.txt"));
```

and that a file can be processed, token by token, in a loop such as:

```
while (filein.hasNext()) {
   String tk = filein.next();
   process(tk); // do something with the token
}
```

(For the wordlist file, a token is simply a word.)

Start your main program by reading the words from words.txt and storing them in a *HashSet<String>*. For the purposes of this program, **convert all words to lower case** before putting them in the set. To make sure that you've read all the words, check the size of the set. (It should be 72875.) You could also use the contains method to check for the presence of some common word in the set.

Checking the Words in a File

Once you have the list of words in a set, it's easy to read the words from a file and check whether each word is in the set. Start by letting the user select a file. You can either let the user type the name of the file or you can use the following method:

```
/**
 * Lets the user select an input file using a standard file
 * selection dialog box.  If the user cancels the dialog
 * without selecting a file, the return value is null.
 */
static File getInputFileNameFromUser() {
   JFileChooser fileDialog = new JFileChooser();
   fileDialog.setDialogTitle("Select File for Input");
   int option = fileDialog.showOpenDialog(null);
   if (option != JFileChooser.APPROVE_OPTION)
      return null;
   else
      return fileDialog.getSelectedFile();
}
```

Use a Scanner to read the words from the selected file. In order to skip over any non-letter characters in the file, you can use the following command just after creating the scanner (where in is the variable name for the scanner):

```
in.useDelimiter("[^a-zA-Z]+");
```

(In this statement, "[^a-zA-Z]+" is a regular expression that matches any sequence of one or more non-letter characters. This essentially makes the scanner treat any non-letter the way it would ordinarily treat a space.)
You can then go through the file, read each word (converting it to lower case) and check whether the set contains the word. At this point, just print out any word that you find that is not in the dictionary.

Providing a List of Possible Correct Spellings

A spell checker shouldn't just tell you what words are misspelled -- it should also give you a list of possible correct spellings for that word. Write a method

```
static TreeSet corrections(String badWord, HashSet dictionary)
```

that creates and returns a *TreeSet<String>* containing variations on *badWord* that are contained in the dictionary. In your main program, when you find a word that is not in the set of legal words, pass that word to this method (along with the set). Take the return value and output any words that it contains; these are the suggested correct spellings of the misspelled word. Here, for example, is part of the output from a sample program when it was run with the HTML source of this page as input:

```
html: (no suggestions)
cpsc: (no suggestions)
hashset: hash set
treeset: tree set
cvs: cs, vs
isempty: is empty
href: ref
txt: tat, tet, text, tit, tot, tut
filein: file in
pre: are, ere, ire, ore, pare, pee, per, pie, poe, pore, prep, pres,
    prey, pro, pry, pure, pyre, re
hasnext: has next
wordlist: word list
getinputfilenamefromuser: (no suggestions)
jfilechooser: (no suggestions)
filedialog: file dialog
```

```
setdialogtitle: (no suggestions)
int: ant, dint, hint, in, ina, inc, ind, ink, inn, ins, inti, into,
     it, lint, mint, nit, pint, tint
```

Note that the program was written so that it will not output the same misspelled word more than once. (This is done by keeping a set of misspelled words that have been output.) If the *corrections()* method returns an empty set, the program outputs the message "(no suggestions)". Since the corrections are stored in a tree set, they are automatically printed out in alphabetical order with no repeats.

The possible corrections that the program considers are as follows:
• Delete any one of the letters from the misspelled word.
• Change any letter in the misspelled word to any other letter.
• Insert any letter at any point in the misspelled word.
• Swap any two neighboring characters in the misspelled word.
• Insert a space at any point in the misspelled word (and check that both of the words that are produced are in the dictionary)

For constructing the possible corrections, you will have to make extensive use of substrings. If w is a string, then *w.substring(0,i)* is the string consisting of the first i characters in w (not including the character in position i, which would be character number *i+1*). And *w.substring(i)* consists of the characters of w from position i through the end of the string. For example, if *ch* is a character, then you can change the *i-th* character of w to ch with the statement:

```
String s = w.substring(0,i) + ch + w.substring(i+1);
```

Also, you will find it convenient to use a *for* loop in which the loop control variable is a *char*:

```
for (char ch = 'a'; ch <= 'z'; ch++) { ...
```

Last modified: Wednesday, 13 May 2020, 3:49 PM