CSE 223 Programming Assignment 2: Strings, Hashes, Linked Lists and Classes Inside Classes

April 28, 2020

Part I

Introduction

This assignment is significantly more complex that Programming Assignment 1, yet the actual amount of code you need to write may be surprisingly small. Again: work methodically and **ask questions along the way**.

The biggest challenge here is thinking of objects as self-contained entities that include data and functions for interacting with that data. Once you come to terms with this, the code starts to almost write itself!

This also won't happen in a day: it will take multiple coding sessions for your brain to settle into this way of thinking. **Start early!** Starting the project a day or two before it's due won't suffice, and you'll suffer throughout the reminder of the class if you do so.

Do not use Eclipse, IntelliJ, or any other IDEs: work from the command line on the Linux server (or your own Linux environment). This will ensure you know the syntax for defining classes, writing a main method, and so on.

1 Description

For assignment 2, you are going to implement a new Java class named Indexer, which will allow you to read the contents of a text file and create an index showing the position(s) of each word in the file.

2 Details

Your class must be named Indexer. You are only responsible for writing this class, but you'll want to develop a main program (or several programs) to help you test it.

Constructor

The class should have a single constructor:

```
Indexer ind=new Indexer();
```

which does nothing in particular!

Public Methods

You must provide the following public methods:

```
public boolean processFile(String filename);
```

Usually the first method you would call after constructing an Indexer. Opens the given filename, reads its contents and, **while reading**, builds an index. Returns true if the process succeeds, false if it fails (typically because the file cannot be read). Note that this method never prints anything (including error messages).

Once you've built an index, you can use the following methods to examine it:

```
public int numberOfInstances(String word);
```

If processFile() has not yet been **successfully** run, return -1. Otherwise, return the number of times that the given word appears in the processed file. If the word does not appear, return 0.

```
public int locationOf(String word, int instanceNum);
```

If n is the value of instanceNum, then this method returns the location of the n^{th} occurrence of word (**NOTE** the first occurrence is 0; the second is 1; and so on). If processFile() has not yet been **successfully** run, return -1. If word does not appear in the processed file, return -1. If instanceNum is too small (<0) or too large ($\geq$the number of occurrences of word), return -1.

```
public int numberOfWords();
```

Return the total number of unique words in the index (-1 if processFile() has not yet been **successfully** run).

```
public String toString();
```

Return the toString value of your hash table. If processFile() has not yet been **successfully** run, return null

Private Methods

You should also implement the following **private** methods:

```
private String cleanupWord(String word);
```

Remove all characters from word that are not letters, according to Character.isLetter(). Return an upper-case version of the resulting string (which could be empty, if none of its original characters were letters).

```
private void addReference(String word, int location);
```

Add location to the end of word's linked list. If word is not presently in the hash, add it first (with an empty linked list). If no file has been processed, do nothing (don't throw an exception!)

Fields

You need one private field in your class, named index. This should be an object of type HashMap.

3 Implementation

To process a file:

- open it using Scanner (if an error occurs, immediately return false);
- construct a HashMap named index. Keys should be Strings, and values should be LinkedLists of Integers;
- initialize a position counter to 0
- read words using hasNext() and next();
- clean the word using cleanupWord();
- *if the word is not empty*, **increment the position counter** and then call your addReference() method (pass it the word and the position counter). addReference() functions as follows:
  - if the word is not in the index, add it along with an empty linked list
  - add the position counter to the end of the linked list
- On end of file, close the Scanner and return true.

cleanupWord() can be a single for loop that builds a return string based in a character-by-character examination of the incoming string. Look at String.charAt() and Character.isLetter().

The remaining methods are mostly just wrappers to calls to other methods.

4 Testing

Here are a sample main program, test file and expected output that can be used to test your Indexer (but make sure you do much more testing than this!)

Sample Code

(This sample program is available on the linux server at /tmp/PA2Test.java; the testfile is at /tmp/testfile.txt Copy these to your own directory to use them.)

```
//
// Test code for Indexer class
// njm
//

public class Main
{
  public static void main(String[] args)
  {
    Indexer ind=new Indexer();
// make sure things don't blow up
    System.out.println("Before processing, ind="+ind);
    System.out.println("Before processing, numberOfInstances returns "+
                       ind.numberOfInstances("TEST"));
    System.out.println("Before processing, locationOf returns "+
                       ind.locationOf("TEST",1));
    System.out.println("Before processing, numberOfWords returns "+
                       ind.numberOfWords());

// process a non-existent file
    boolean status=ind.processFile("badfile.bad");

    System.out.println("\nAfter processing bad file, status="+status);
    System.out.println("\nAfter processing bad file, ind="+ind);
    System.out.println("After processing bad file, numberOfInstances returns "+
                       ind.numberOfInstances("TEST"));
    System.out.println("After processing bad file, locationOf returns "+
                       ind.locationOf("TEST",1));
    System.out.println("After processing bad file, numberOfWords returns "+
                       ind.numberOfWords());

// no process a good testfile
    status=ind.processFile("testfile.txt");

    System.out.println("\nAfter processing good file, status="+status);
    System.out.println("\nAfter processing good file, ind="+ind);
    System.out.println("# of instances of BADBADBAD="+
                       ind.numberOfInstances("BADBADBAD"));
    System.out.println("# of instance of TEST="+
                       ind.numberOfInstances("TEST")+"\n");
// iterate over all instances (plus a few out of range)
    for (int i=-1;i<=ind.numberOfInstances("TEST");i++){
      System.out.println("Instance " + i + " is at location "+
                         ind.locationOf("TEST",i));
    }
    System.out.println("\n# of unique words="+ind.numberOfWords());
  }
}
```

testfile.txt

```
This, is a test! So it is!!!, so it is,
and so, I say, yes? or no! Time to test this.
```

```
What will we do? :)  Test test test
Yes! Goodbye!
```

Corresponding Output (note that "THIS" occurs at position 1, not 0)

```
Before processing, ind=null
Before processing, numberOfInstances returns -1
Before processing, locationOf returns -1
Before processing, numberOfWords returns -1

After processing bad file, status=false

After processing bad file, ind=null
After processing bad file, numberOfInstances returns -1
After processing bad file, locationOf returns -1
After processing bad file, numberOfWords returns -1

After processing good file, status=true

After processing good file, ind={A=[3], NO=[17], OR=[16],
YES=[15, 29], I=[13], IS=[2, 7, 10], SAY=[14], TIME=[18],
IT=[6, 9], DO=[25], WHAT=[22], WE=[24], GOODBYE=[30],
TEST=[4, 20, 26, 27, 28], AND=[11], THIS=[1, 21], WILL=[23],
TO=[19], SO=[5, 8, 12]}
# of instances of BADBADBAD=0
# of instance of TEST=5

Instance -1 is at location -1
Instance 0 is at location 4
Instance 1 is at location 20
Instance 2 is at location 26
Instance 3 is at location 27
Instance 4 is at location 28
Instance 5 is at location -1

# of unique words=19
```

5 Grading

Your assignment must be downloadable from GITLab by me in order to be graded. I will download using the command:

```
git clone git@gitlab.com:yourGITid/CSE223PA2.git
```

You can mimic this yourself to confirm that your repository is located and named correctly. There's no sure way to check that you have added me as a reporter: just make sure you do!

Points are awarded as follows:

- Project can be downloaded and compiled: 20 points
- Indexer class defined: 5 points
- processFile ingests the given file and stores some cleaned words in a HashMap: 10 points
- processFile associates a LinkedList containing at least one location with (some) words: 5 points
- processFile saves all words and a LinkedList with some locations: 5 points
- processFile saves all words and a LinkedList with all locations: 5 points
- toString, numberOfWords, numberOfInstances and locationOf: 20 points (5 points each if perfect)
- correct handling of uninitialized index(processFile not successfully run): 5 points
- required private methods implemented: 10 points
- style: 15 points if you have **all of the following**:
  - consistent indenting, placement of braces, etc.
  - all code commented (**every line** unless it's extremely obvious why that line is there)
  - each method described by an introductory block of comments.
  - entire class described by a block of comments in the beginning (in your own words).

6 Submission

Create a repository named CSE223PA2 on GITLab. The repository should contain Indexer.java **in the top level of the repository**. You can include any other files, subdirectories, etc. as you like (I will ignore them). Add me (nickmacias) as a reporter on your project.