Weblog Processor

Tim Storer
Assessed Exercise 1, Weight 10%

- Due at 10:00 am on the day after your lab during the week of 15th February.
- Reminder: credit for OOSE2 will be refused unless you submit at least one of the two assessed exercises and attend at least three OOSE2 lab sessions.

This handout describes the first assessed exercise for OOSE2. Read it over carefully. If you do not fully understand what you need to do, please discuss with your tutor during your lab the week of 1st February.

You are responsible for modifying an existing, working application to meet a set of modified or additional requirements. You have been provided with the requirements specification and a set of test JUnit test cases that are used to validate the classes that make up the application. You have been given java files for all of the classes that constitute the application. Additionally, you have been provided with the html files generated by javadoc.

1 Introduction

The application `LogStats` processes log files from web servers. For each request received, the web server writes a line in its log file consisting of a date, in the format `dd/mm/yyyy`, followed by white space, followed by the domain name of the machine that submitted the request; a domain name consists of a sequence of .-separated fields, such as dcs.gla.ac.uk; the last such field is termed a country code.

For a given period marked by begin and end dates, `LogStats`, keeps track of the number of requests received from each country code. For example, assume that the begin date is 01/01/2001 and the end date is 31/12/2001; `LogStats` then processes a log file, keeping only log entries associated with dates 01/01/2001 to 31/12/2001, inclusive. If the following log file were processed by `LogStats` with these dates,

```
02/03/2001 a.b.c.d
31/12/2000 g.h.i.d
15/06/2001 x.y.z.q
01/01/2002 x.y.b.q
28/09/2001 r.s.q
```

it would ignore the 2nd and 4th lines (outside of the period); of the remaining lines, it would indicate that it had seen one log line in the period associated with country code “d”, and two log lines associated with country code “q”.

Several alterations to the LogStats application are necessary. In each case, the change should be documented by revising the appropriate test case (as well as the requirements document) and then be corrected in the implementation.

1. The current program does not exactly meet the requirements specified in the requirements document: it is required that each date be validated (see requirements document). The current implementation of date validation does not meet the requirements.

2. The current program prints out the entries from smallest count to highest count; for country codes possessing identical counts, the entries are printed out in alphabetic order of country code. The customers are unanimous in demanding that the entries be printed out in the reverse order i.e. from highest count to lowest count; for country codes with the same count, you should continue to output them in alphabetic order by country code.

3. Finally, customers that have very popular web sites have indicated that the program runs EXTREMELY slowly when processing large log files with entries from thousands of different machines. This is probably due to the way that the country codes are stored in a `CountryList`. You need to track down the cause of the problem, document it as a test method in `TimedLogStatsTest` and implement any required modifications.

2 Approach

The most important first step is to thoroughly read the requirements document to make sure you understand it. Then study the java classes and test cases; it is critical that you understand what each of the methods are doing AND how they are performing their tasks. Only when this is the case are you in a position to make the required modifications.

1. First you must understand why the validation that occurs in `Date` does not meet the requirements. Recall that a leap year satisfies the following constraints:

   `year % 4 == 0`, unless `year % 100 == 0`, except if `year % 400 == 0`

   for example, 2008 is a leap year, 1900 was not a leap year, 2000 was a leap year.

2. You must understand how `sortByCount()` works in `CountryList`; you could modify `sortByCount()` to sort by decreasing count, or you could add a new method that performs such a sort by decreasing count.

3. You must understand how the system computation time varies with the number of entries processed AND the number of unique country codes. A JUnit test harness, `TimedLogStatsTest` , is provided to assist you in determining the behaviour or `LogStats` as a function of the number of entries processed and the number of unique country codes.

   The `TimedLogStatsTest.testInsertionTiming()` method times the amount of time taken for a varying (large) number of entries to be inserted into the `CountryList`. This is done with the aid of two utility methods:

   - `generateLargeLogInputStream(nccodes, nentries)` returns an input stream representing a large log file. `nentries` specifies the number of lines in the log file, while `nccodes` specifies the number of country codes to be used. Invoking `doLargeCountryInsert(1000,1000)`, for example will generate 1000 lines of log entries on the `InputStream`, each with a unique country code.

- `doLargeCountryInsert()` produces timing statistics on stdout (the Console in Eclipse) for a single insertion run.

To test the scaling behaviour of `LogStats` with the number of unique country codes, you need to modify the test case to report timings for a range of different log files sizes. You might use `doLargeCountryInsert(int,int)` for the following values:

```
1000 1000
2000 2000
4000 4000
8000 8000
16000 16000
32000 32000
```

If you generate results like those I describe above, using start date of 01/01/1900 and end date of 01/2/2008, you will obtain the elapsed time as a function of the number of unique country codes; if you plot the elapsed time as a function of the number of unique country codes, you should see a pattern emerging. You must determine what is causing this pattern, and design changes to the program to eliminate the pattern. As always, before you change any classes, you must modify the appropriate test harnesses. You should also generate a set of results in which the number of unique country codes remains constant, but the number of entries in the log file vary to see if these give any indication as to the cause of the slowdown experienced by customers.

3 Set Up

When you set up Exercise1 using AMS you will obtain an Exercise1 folder. For this assignment, you will have to switch your Eclipse workspace to the Exercise1 folder.

Create a new Java project, named OOSE2-Ex1, creating it from the existing folder Exercise1. Make sure that the "create separate folders for source and class files" radio button is checked. At this point, all of the files in the folder tree descending from Exercise1 will become part of your project in Eclipse. These files include the source code and javadoc-generated documentation for the classes and applications. The requirements document is available in the /docs folder.

You are to make your modifications to the files in place. It is a good idea to save copies of these original files before you begin to modify them, just in case you make a mistake. Systems like CVS or Subversion are available in Eclipse to support this; if you are not familiar with such systems, simply making a copy should be sufficient protection. I suggest that if you are saving a copy of "name.ext" that you name the copy "name.ext.save".

Note: do NOT rename any of the `.java` files in `Exercise1/src`, any of the `.html` files in `Exercise1/docs/api/`, nor any of the `.doc` files in `Exercise1/docs`.

Open `LogStats.java` in Eclipse. `LogStats` is a console application and takes command arguments. In order to familiarise yourself with the `LogStats` application, you can run the application from the console. Note, that any testing *must* be performed using the JUnit test case suite. To run `LogStats` from the console, choose the "Open Run Dialog..." item in the "Run" menu; you will see a tab labelled "(x)= Arguments". If you click on that tab, you will see a box labelled "Program arguments:". There is a sample log file in the oose2 package. Type the following into the "Program arguments:" box.

```
01/01/1902 02/02/2008 logs/small.log
```

This should produce the following output in the console:

```
Statistics for oose2/small.log in the date interval [01/01/1902, 02/02/2008]
edu        4
fr         4
it         4
com        8
de         8
uk        20
```

4 File list in directory Exercise1

- `doc/*.doc` the requirements document
- `docs/api/` javadoc files for the classes and programs
- `src/oose2/ex1b/*.java` source files for the classes
- `src/oose2/ex1b/tests/*.java` source files for the test cases
- `logs/*.log` log files for testing.
- `logs/small-2001.01.01-2002.12.31.out` correct output when LogStats is invoked on `logs/small.log` with:
  - begin date = 01/01/2001
  - end date = 12/31/2002

5 What you will be submitting

When you submit your solution to Exercise 1, we will copy the following files and directories into the AMS system:

- Exercise1/docs/requirements.doc
- Exercise1/docs/api/ - API documentation.
- Exercise1/src/oose2/ex1b/*.java all classes that have been modified
- Exercise1/src/oose2/ex1b/tests/*.java all test cases that have been modified

Any other files that you create anywhere in the Exercise1 tree will not be copied.

6 Assessment

Your submission will be marked on a basis of 10 marks for the assignment. Successfully addressing the Date validation bug will be worth 2 mark, the reverse ordering of the output will be worth 3 marks, and the performance fix will be worth 5 marks. Note that addressing the changes requires revision to the accompanying documentation, as well as both a revised test suite (which you should do first) and a revised implementation.