CS1003 Week 3 Practical: File Processing

This practical is worth **20%** of the overall coursework mark. It is due at **9pm** on **Thursday 14th February (week 3)**. As for every practical, you should arrive in the lab having prepared in advance, by studying this specification and reviewing relevant course material. If you don't do this you will waste time during your lab sessions.

Skills and Competencies

- developing robust software to manipulate data stored in files
- choosing an appropriate way to represent data in memory
- choosing appropriate classes and methods from a file API
- identifying and dealing with possible error conditions
- testing and debugging
- writing clear, tidy, consistent and understandable code

Setting Up

On this module we intend to use the automated checker `stacscheck` for most assignments. You should already be familiar with using it from CS1002 last semester. As you know, using the automated checker means that we have to impose some restrictions on naming of Java files, Java classes and where to put your files. For this practical, your program must have a `main` method in a class called `W03Practical` which is in a file `W03Practical.java` that is located inside your `src` directory in your `W03-Practical` directory.

The following steps are provided to help you set up your practical directories in such a way that they will be compatible with the automated checker. These steps are similar to those you carried out last semester on CS1002 at the start of practicals:

- Log in to a machine booted to Linux
- Check your email in case of any late announcements regarding the practical.
- Launch ***Terminal***.
- If you do not already have a *cs1003* directory for your CS1003 coursework, you can use the command `mkdir cs1003` to create it and then move to it using the command `cd cs1003`.
- Create a new directory called *W03-Practical* inside your *cs1003* directory and move to it by using the commands `mkdir W03-Practical` and `cd W03-Practical` for example.
- Create a new directory called *src* inside the *W03-Practical* directory to save your java source files using `mkdir src`. All your java files must be stored within this directory.
- Use appropriate `cd` commands (such as `cd src`) to navigate to the *src* directory within the *W03-Practical* directory.
- Create a new Java file *W03Practical.java* using e.g. `touch W03Practical.java`.
- Using *jEdit*, declare the class `W03Practical` and add an empty `main` method. Now save your program. You can also use your favourite IDE if you wish, as long as the files are named and stored as indicated above.
- Your program should compile using `javac *.java` at the command line, and run with `java W03Practical`. Please make sure this works, even though it won't do anything exciting yet.
- Create a new *LibreOffice Writer* document for your report and save it as *W03-Practical-Report*, inside your *W03-Practical* directory add an appropriate header at the top such as

  CS1003 W03 Practical 012345678 13 Feb 2019

- If you think you need more detailed instructions on how to use the Unix command line, have a look at last semester's CS1002 instructions for the Week 1 practical at:
  https://studres.cs.st-andrews.ac.uk/CS1002/Practicals/W01/CS1002-W01-Practical-Introduction.pdf

Requirements

Write a Java program to process data from a file. For this practical, the data is stored in comma-separated-values (CSV) format. Your program needs to read in data from a given file, perform some processing, and create a new file containing the results. The provided data contains customer reviews and ratings for restaurants in 31 European cities, sourced from TripAdvisor and distributed on Kaggle[1].

The format of the input data is illustrated below. The first line is the header line, and the remaining two blocks are example lines for two data records. Each data record appears on a separate line in the file.

```
Number,Name,City,Cuisine Style,Ranking,Rating,Price Range,Number of Reviews,Reviews,URL_TA,ID_TA

0,Martine of Martine's Table,Amsterdam,['French'; 'Dutch'; 'European'],1.0,5.0,$$ -
$$$,136.0,[['Just like home'; 'A Warm Welcome to Wintry Amsterdam']; ['01/03/2018';
'01/01/2018']],/Restaurant_Review-g188590-d11752080-Reviews-Martine_of_Martine_s_Table-
Amsterdam_North_Holland_Province.html,d11752080

1,De Silveren Spiegel,Amsterdam,['Dutch'; 'European'; 'Vegetarian Friendly'; 'Gluten Free
Options'],2.0,4.5,$$$$,812.0,[['Great food and staff'; 'just perfect']; ['01/06/2018';
'01/04/2018']],/Restaurant_Review-g188590-d693419-Reviews-De_Silveren_Spiegel-
Amsterdam_North_Holland_Province.html,d693419
```

The first column in the data is a *Restaurant Number*, a numbering of restaurants within each city. That is, each city will have a restaurant with number 0 which can be treated as an `int` type. The basic requirements do not require you to deal with rounding errors (using a simple `double` type for *Ranking*, *Rating*, and *Number of Reviews* attributes is fine). You can simply treat the remaining attributes as Strings. You can also assume that data is ordered according to *City* (and by the un-named *Restaurant Number* column within each city).

Various versions of the file, of differing lengths, can be downloaded from:

- https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/W03/data/clean_small.csv
- https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/W03/data/clean_large.csv

The Task

You are to write a Java program which will read and process the data (indicated above) and write output to an output file. The overall operation of your program should be as follows:

- The program should be written to expect and use *two* command-line arguments (by making use of the `args` parameter to your `main` method in your `W03Practical` class).
  - `args[0]` should represent the path of the *input file*.
  - `args[1]` should represent the path to the *output file* where your program will write its output.
  - `args[2]` should represent the *Minimum Rating* to use when processing restaurants.
- If the command-line arguments are not supplied, then your program should print to `System.out` the line containing usage information as shown below
- Your program should process the *input file* and create an *output file* containing:
  - for each *City*, the total number of restaurants
  - for each *City*, the total number of restaurants with a rating *greater or equal* to and *less than* the specified *Minimum Rating* criterion, and the percentage of restaurants with a rating *greater or equal* to the specified *Minimum Rating* criterion,
  - a summary at the end of the output file (over all cities) consisting of the total number of restaurants with a rating *greater or equal* to and *less than* the specified *Minimum Rating*

[1] Obtained and adapted from: https://www.kaggle.com/damienbeneschi/krakow-ta-restaurans-data-raw

criterion, and the percentage of restaurants with a rating *greater or equal* to the specified *Minimum Rating* criterion.

- All numbers should be printed with up to 1 decimal place. You may use the `DecimalFormat`[2] class to achieve this.

Below are some examples of the expected output when executing your program from the `src` directory and for some different command-line arguments indicating the functionality that you should provide:

```
java W03Practical
Usage: java W03Practical <input_file> <output_file> <minimum_rating>

java W03Practical /cs/studres/CS1003/Practicals/W03/data/clean_small.csv ../output.txt 4.0
```

The first invocation above does not pass any command-line arguments to the program and results in the required usage message being displayed in the terminal window. The second invocation does pass the expected command-line arguments to the program and as a result does not produce any console output to the terminal. Instead, the program writes the required information to the file `output.txt` in the `W03-Practical` directory as shown below (assuming the program is executed from the `src` directory that is inside your `W03-Practical` directory).

Content of file `output.txt` for second invocation above:

```
Amsterdam has 5 restaurants.
Number of restaurants with rating above (or equal to) 4: 5
Number of restaurants with rating below 4: 0
Percentage of restaurants with rating above (or equal to) 4: 100%

Athens has 5 restaurants.
Number of restaurants with rating above (or equal to) 4: 4
Number of restaurants with rating below 4: 1
Percentage of restaurants with rating above (or equal to) 4: 80%

Overall
Number of cities: 2
Number of restaurants: 10
Number of restaurants with rating above (or equal to) 4: 9
Number of restaurants with rating below 4: 1
Percentage of restaurants with rating above (or equal to) 4: 90%
```

The program should produce the output exactly as shown above and such output will be expected by the automated checker. Your program should deal gracefully with possible errors such as the input file being unavailable, or the file containing data in an unexpected format.

Running the Automated Checker

As usual, please use the automated checker to help test your attempt. The checker is invoked from the command line on the Linux Lab Machines in your `W03-Practical` folder:

```
stacscheck /cs/studres/CS1003/Practicals/W03/Tests
```

If all goes well, you should see something similar to below

```
Testing CS1003 Week 3 Practical
- Looking for submission in a directory called 'src': found in current directory
* BUILD TEST - basic/build : pass
* COMPARISON TEST - basic/Test01_args/01_no_arguments/prog-expected.out : pass
* COMPARISON TEST - basic/Test01_args/02_2args/prog-expected.out : pass
* COMPARISON TEST - basic/Test01_args/03_nan/prog-expected.out : pass
* COMPARISON TEST - basic/Test02_small/small_00/prog-prog.out : pass
```

[2] https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html

```
* COMPARISON TEST - basic/Test02_small/small_30/prog-prog.out : pass
* COMPARISON TEST - basic/Test02_small/small_35/prog-prog.out : pass
* COMPARISON TEST - basic/Test02_small/small_40/prog-prog.out : pass
* COMPARISON TEST - basic/Test02_small/small_50/prog-prog.out : pass
* COMPARISON TEST - basic/Test03_large/large_00/prog-prog.out : pass
* COMPARISON TEST - basic/Test03_large/large_30/prog-prog.out : pass
* COMPARISON TEST - basic/Test03_large/large_35/prog-prog.out : pass
* COMPARISON TEST - basic/Test03_large/large_40/prog-prog.out : pass
* COMPARISON TEST - basic/Test03_large/large_45/prog-prog.out : pass
* COMPARISON TEST - basic/Test03_large/large_50/prog-prog.out : pass
* INFO - basic/TestQ_CheckStyle/infoCheckStyle : pass
--- submission output ---
Starting audit...
Audit done.
---
17 out of 17 tests passed
```

If it is not going so well, here are some things to consider

- If the automated checker cannot be started, then you have most likely mistyped the commands to invoke the checker shown above.
- If the automated checker runs but fails at the build stage, then no other tests can be conducted, so you will have to fix this issue first. Likely reasons for the build failure include: executing the checker from some directory other than your `W03-Practical` directory, specifying an incorrect path to the tests, and your program containing errors such that it cannot be compiled.
- If *Test01* fails, it is because your program is not producing the expected output when no command-line parameters are supplied on execution. Remember to check the length of the `args` array and print a suitable message if it too small.
- If some or all of *Test02 – Test03* fail, then your program is most likely not writing the expected output to the file that is specified on the command line when executing your program. Make sure that you use the `args` array elements correctly in your code.
    - Each of these tests will run your program with specified input files and output files. If you see a line such as:

      ```
      --- submission stderr output ---
      cat: prog-output.txt: No such file or directory
      ```

      your program has not written to the output file specified as a command-line parameter.
    - Failing these tests could be down to logical errors. Check the logic and debug your program to spot errors and fix them. Try running the automated checker again to verify your fix.
- Please make sure you comment out all your debugging print statements before running the automated checker.
- The final test *TestQ CheckStyle* runs a program called Checkstyle over your source code and uses a style adapted from our St Andrews coding style (informally known as the *Kirby Style*). You may receive a lot of output from the style checker for your program. In order to address these, you can look at the published guide at

  https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/programming-style.html

  The style recommends the use of spaces as opposed to TABs for indentation. Don't worry too much about this.

If nothing is working and you don't know why, please don't suffer in silence, ask one of the demonstrators in the lab. The automated checking system will help you with some basic testing of your program, and you can document your success (or otherwise) with it in the Testing section of your report.

Testing

You are encouraged to create and document your own tests to test how your program deals gracefully with possible errors such as the input file being unavailable, or the file containing data in an unexpected format. To prove that you have tested the above scenarios, paste terminal output from each of your tests into your report in the Testing section. Be clear what each test demonstrates.

Deliverables

Hand in via MMS to the *Week 3 Practical* slot, as single `.zip` file containing your entire `W03-Practical` directory which should contain:

- All your Java source files as specified above
- A report containing the usual sections for *Overview*, *Design & Implementation*, *Testing*, *Examples* (example program runs), *Evaluation* (against requirements), *Conclusions* (your achievements, difficulties and solutions, and what you would have done given more time). You can use any software you like to write your report, but your submitted version must be in PDF format.

Extensions

Please note: It is key that you focus on getting a very good solution to the basic requirements prior to touching any of the extension activities. A weak solution with an attempt at extensions is still a weak solution. Once you have a very good solution to the basic requirements, you may wish to experiment further and could try any or all of the following:

- Alter your program to take additional command-line parameters which act as switches to e.g.:
  - Output
    - the number of restaurants in each city for each *Cuisine Style*
    - the name and rating of the best rated restaurants for each cuisine style in each city
  - Display some aspect of the data in a graphical format.
  - Make your output file into a web page.
- Alter your program to handle the original data from the site containing commas within CSV fields (escaped using double quotes) as included in

  https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/W03/data/original_large.csv

Don't forget to document your extension work in your report.

Marking

Your submission will be marked using the standard mark descriptors in the School Student Handbook:

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

However, more specifically, for this practical:

- 1-6: Very little evidence of work, software does not compile or run, or crashes before doing any useful work. You should seek help from your tutor.
- 7-10: A decent attempt which gets part of the way toward a solution, but has serious problems such as not compiling (at the low end), or lacking in robustness and maybe sometimes crashing during execution (at the high end).

- 11-13: A solution which is mostly correct and goes some way towards fulfilling basic requirements, but has issues such as not always printing out the correct values.
- 14-15: A correct solution accompanied by a good report, but which can be improved in terms of code quality, for example: poor method/OO structure, lack of comments, or lack of reuse.
- 16-17: A very good, correct solution to the main problem containing clear and well-structured code achieving all required basic functionality, demonstrating very good design and code quality, good method and class decomposition, elegant implementation of methods with reuse where appropriate, and accompanied by a well-written report.
- 18-19: As above, but implementing at least one or more extensions, accompanied by an excellent report. However, as mentioned above, it is key that you focus on getting a very good solution to the basic requirements prior to touching any of the extension activities. A weak solution with an attempt at extensions is still a weak solution.
- 20: As above, but with multiple extensions, outstanding code decomposition and design, and accompanied by an exceptional report.

Lateness

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good Academic Practice

The University policy on Good Academic Practice applies:

http://www.st-andrews.ac.uk/students/rules/academicpractice/

Hints

Rather than trying to develop a complete solution in one go, it may be easier to produce a series of versions, starting with something very simple. Here is one possible sequence:

1. Write your program to open the small version of the input data file and prints out its contents to the console, line by line, using calls to `System.out.println()`.
2. Refine the previous version so that it writes the contents of the input data file to a new output file.
3. The `String.split()` method may be useful for splitting up a line of text from the input file into individual strings.
3. Alter the previous version so that it computes summary values (counting the number of restaurants in the city) and writes these to the file for each city as indicated in the sample output above.
5. Refine the previous version so that it also keeps track of the number of restaurants below and greater or equal to the specified Rating threshold.
7. Test your program with missing and badly formatted input files, and add exception handling where necessary to produce informative error messages.

You may wish to make a new class at each stage, so that you can easily go back to a previous stage if you run into problems.

Prior to submitting, please make sure that your most recent version of the program can be compiled and run using your `W03Practical` class.