C1768263 Simple Weather Data Viewer

COURSEWORK SUBMITTED
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MSC COMPUTING

APRIL 2018

Report

Introduction

The aim of this project was to implement a *"Simple Weather Data Viewer".*

Summary

The author has successfully developed a program in Java according to the following criteria:

1. The application should read data stored within the attached CSV files and present that data in a *"suitable format".*
2. The application should allow individual weather stations to be selected, which provide the user with more information… including *"suitable basic charts"* presenting the data.
3. It should allow users to generate a report containing a weather forecast for all stations.
4. The application should include *"an appropriate GUI, using JavaFX".*

In this report the author shall demonstrate the program, how it's operated and what techniques were used with regard to the code. The program was written in Java, using IntelliJ, and has been tested in Windows, Mac and Linux environments. The project took approx. 40 hours to complete.

Coding Philosophy

The program, in its current form, is the third attempt at solving this assignment. Initially, and in addition to the course, the author mainly used NetBeans with TMC to run and test code securely. In NetBeans, the author got nearly all of the code working, but without a GUI.

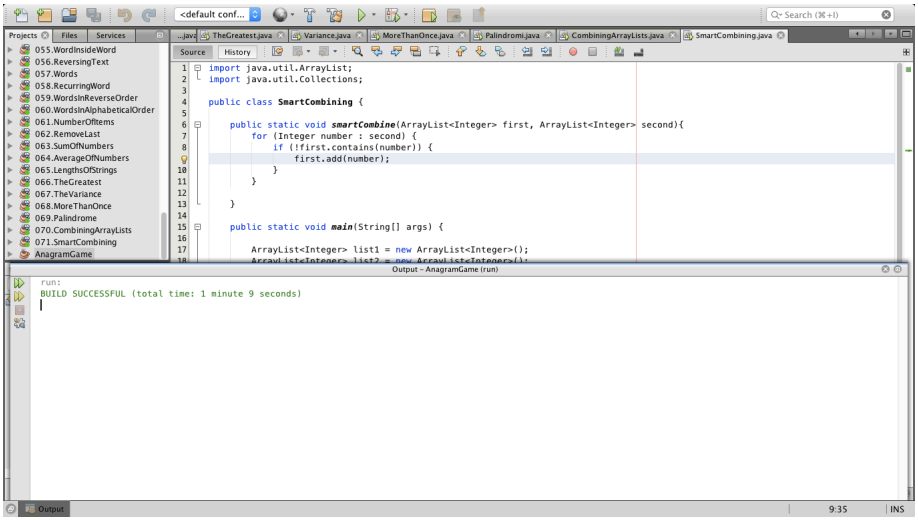

Figure 1 – Java pseudocode in NetBeans (Mac OS)

Up till this point, the author had no real knowledge or experience with JavaFX. After revisiting the notes, and having done some research on Google and YouTube, the author swapped tmcbeans for the IntelliJ IDEA.

None of the code was wasted though, and most of it was copied-and-pasted into the new project directory. *Thankfully*, the author has been meticulous in documenting all of the code, while it's still fresh in his mind. Additionally, the author believes that:

- **Start by doing something simple** – generally, programmers try and break problems down into smaller (easy to solve) tasks. Reading and writing code takes effort, finding one small problem in a very large code block also takes effort. Programmers tend to write as few lines as possible, they use functions, and reuse code wherever possible.
- **If you're stuck, take a break and check Stack Overflow** – there are lots of resources online, and someone has probably already asked the exact same thing. If not, grab a pencil and some paper and try and rethink the problem. Inspiration will come.
- **Name functions and variables properly/consistently**, so you can revisit the code in a couple of days, next week or in six months.
- **Use whitespace and indentation** to keep code clean and well organised.
- ***"There's no such thing as perfection… You just run out of time"*** – **Peter Jackson**. The author thinks it's really important to keep reminding yourself to enjoy coding, and that even if it doesn't work it's not the end of the world.

Are all good coding practices and, where possible, has used them as part of the development process.

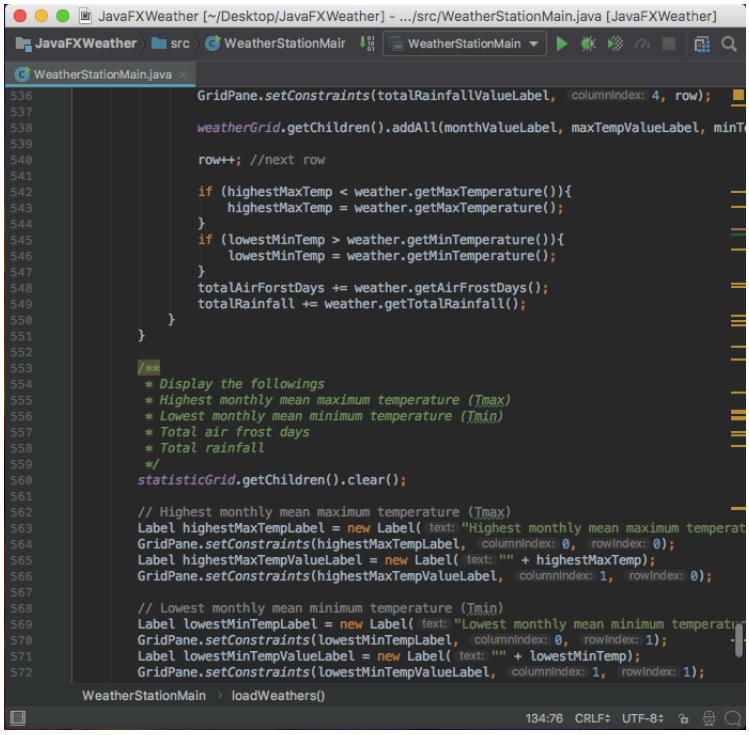

Figure 2 – WeatherStationMain.java in IntelliJ (Mac OS)

Design, Structure

The second iteration program included a simple GUI but, ultimately, the author thought that it was *too* simple. Essentially the program had just the one stage and one scene, with a large grid at the top containing ALL weather data belonging to that station, and a graph at the bottom (with 100+ data points, read line by line).

The third, and final, version of the program has two scenes:

- A grid containing data for that station, that year (both having been selected by the user), and
- A bar chart representing that data.

The main scene is divided into three sections. At the top, there are two dropdown buttons, where users can select the desired weather station and year respectively (2017 by default). Under that, there's a grid containing: the Month, Max Temperature, Min Temperature (°C), Air frost days and Rainfall (mm) as selected above. Months of the year have been hardcoded – i.e., JAN, FEB, MAR not 1, 2, 3. In fact, most values from the CSVs have been edited at one time or another (inside the program) with sting formatting ect., ect. but, ultimately, the author decided against. The author thought that changing these values gave them extra meaning, and that anyone wanting to see raw data could simply open the CSV files directly. Strings (mostly units of measure) where instead placed in the header, to reduce clutter. At the bottom, for convenience: the highest max temperature, lowest min, total air frost days and rainfall for the year are displayed. And, below that, three buttons: *"View Chart"*, *"Generate Report"* and *"Quit Application"*.

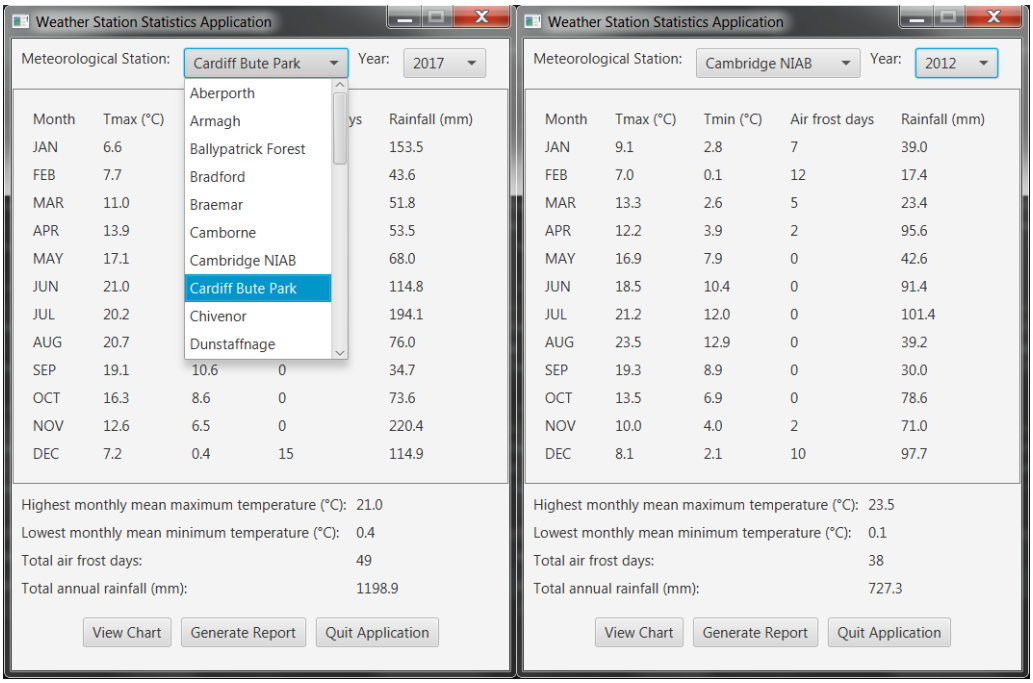

Figure 3 – MainScene (WeatherStationMain.java, Windows 7).
Left: Cardiff Bute Park, 2017. Right: Cambridge NIAB, 2012.

Clicking on *"View Chart"* changes the scene to ChartScene. This scene is essentially one big bar graph, populated by the values selected earlier. The graph is labelled, with months read left to right, at the bottom. Under that, there's a checkbox, where users can add or remove rainfall to the graph, and a *"Back"* button, to return to the main scene. Originally, the rainfall data series **dwarfed** the other values, but the author changed the units from mm to cm, significantly reducing the effect.

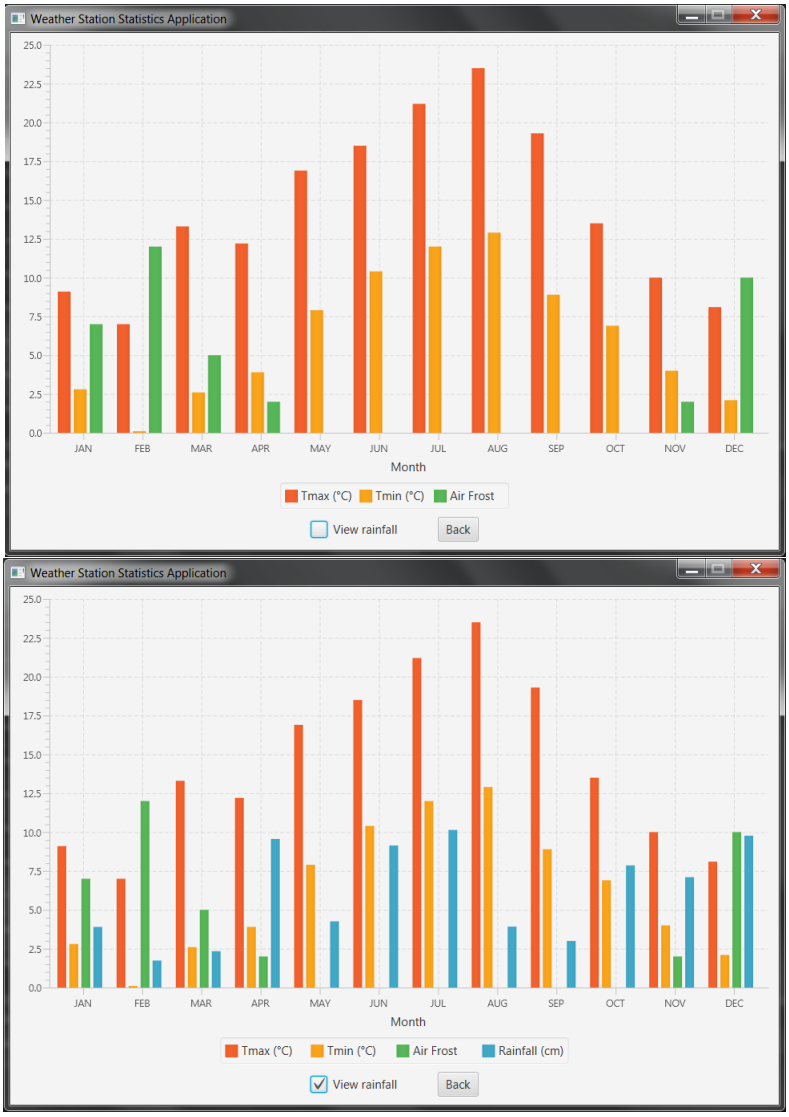

Figure 4 – ChartScene (WeatherStationMain.java, Windows 7).
Top: WITHOUT Rainfall. Bottom: WITH Rainfall.

Note: this is just a new scene, NOT a new stage or window.

Having returned to MainScene, users may decide to create a report (as described in task 3) by clicking on *"Generate Report"*. This action opens the *"Save As"* window, where users can type a file name and select their preferred directory. Files are saved by default under the .txt extension and STRICTLY follow the:

Number: <sequence number>
Station: <station name>
Highest: <month/year with highest tmax>
Lowest: <month/year with lowest tmin>
Average annual af: <average days of frost per year>
Average annual rainfall: <average annual rainfall>

format (see figure 5). Having chosen a file name and location, the window closes and another opens to confirm success.

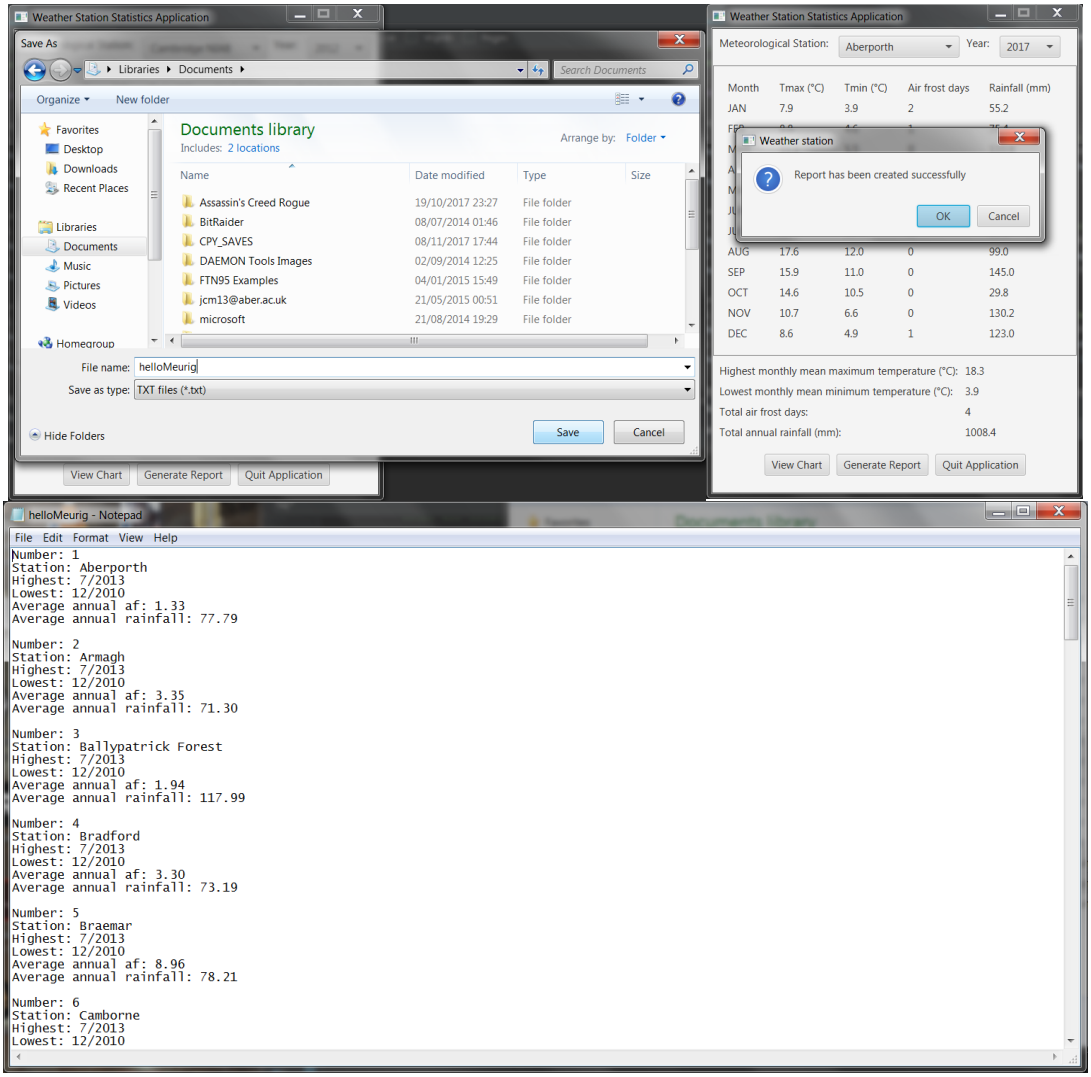

Figure 5 – Save As (left), Confirmation (right) and a Test Report (bottom).

This part of the code is very simple. Using the **bw.write()** command programmers can write just about anything to a text file and, initially, these reports were much more flamboyant: introducing themselves and their structure, as well as welcoming the reader. That said, *again* the author decided against using these features (some of them *I think* still in the code as comments). The author thinks that task 3 is much more specific than the other tasks, and that reports, based on the language in the question, are *perhaps* more likely to be read by another program than by the user.

Design Rationale

The author believes that a simple layout makes data easy to read, navigate and find content, all without having too much on-screen at any one time. The program was designed with smaller screens in mind, like on tablets and smartphones. Also, the size of the window is fixed. The author has done his best to maximise the information displayed on-screen, without just parsing the data and presenting it in table format. The author is quite proud of the result.

Testing

The application has been run successfully in Windows, Mac and Linux, both from home and in the Labs. The program can be launched inside IntelliJ, or by double clicking the WeatherStationMain.java file (SDK: 1.8, language level: 8).