This report is for student s1869672

The raw overall mark is (21/25)

The awarded mark includes a deduction for any days late
(5% per calendar day).

Days late: (0)
The awarded mark is **(21/25)**

As described in the coursework instructions, marks are awarded for two aspects: readability and correctness. See below for comments on these and other aspects of your submission.

1. Readability (10 marks)

Your application should be well-structured and clear. You should consider the readability of your code, thinking that it will be passed on to the members of the air quality research project to extend and maintain as their needs change.

Aspects markers focussed on included: how well you structured your code, the identification of a value-to-colour mapping function, the use of final modifiers when specifying constants, the use of Mapbox geoJSON libraries, and the use of Java 11 local variable type inference.

Readability mark: (6/10)

Comments:

You have a largely unstructured solution. There were opportunitiesto define methods here to structure your application, but you have not chosen to do this. This makes your code more difficult to read and more difficult to maintain.

You implemented the function which maps values to colours as a separate method in your application. This was a good idea because you will be able to re-use this method in your next coursework.

You have not declared constant (final) values in your application; this was a missed opportunity.

You used the Mapbox SDK for GeoJSON to help in generating your heatmap output. This was a good decision because this SDK has many useful methods for handling GeoJSON.

You have made a lot of use of local variable type inference in your
application; this is good practice. We were pleased to see this.

2. Correctness (15 marks)

Your application should correctly render the heatmap corresponding to
the predicted data values in the input text file.

Correctness mark: (15/15)

Correctness checking involved first using maven and your pom.xml file
to compile your code and generate a jar Java archive file containing
your compiled classes along with those of Java libraries on which your
code depended.

Then we ran your code on a sample input file and used an auto-marking
program to assess the quality of the geoJSON output generated by your
program. The following sections give details on any issues that came
up in the build process and in running your code. They also describe
attached files that contain further information.

The correctness mark is based on a combination of
- the quality of the build (the compilation and packaging) of your submission,
- how precisely instructions for the input and output files were followed,
- the score assigned to your geoJSON output.

3. Top-level structure of submitted zip file

We expected the zip file to include a directory named "heatmap"
containing files and directories arranged as required by the maven
build tool. In particular, we expected to find your pom.xml file in
the heatmap/ directory and your Java files in directories below the
heatmap/src/main/java sub-directory. If we did not find this, comments
are made here. No marks were deducted for unexpected top-level
structures.

4. Java packages

We expected your code to be in the package "uk.ac.ed.inf.heatmap" and
perhaps sub-packages of this, and the path(s) to your Java files to
correspond to their package(s). If this was not the case, we comment
on this here. No marks were deducted for use of unexpected packages.

Alternate package (if not uk.ac.ed.inf.heatmap):

5. Compilation, packaging and running

Remarks on issues encountered (blank if none):

6. Input and output files

The instructions required that the name of the input file in the
working directory be provided as an input argument. Some submissions
instead only looked for a file with a fixed name such as
"predictions.txt" and some looked for in locations other than the
working directory.

Also the instructions required that the output file be named
"heatmap.geojson" and be written to the working directory. Some
submissions instead wrote to files with different names, wrote the
file in directories other than the working directory, or wrote to
standard output rather than writing to a file.

Issues encountered and addressed (blank if none):

7. Input file formatting

The instructions specified that the input file should consist of 10
lines of 10 numbers, with numbers on each line separated by commas.
The best submissions allowed for space characters before and after the
comma characters, and were insensitive to the kind of newline
character or character sequence used to terminate input lines.

The standard under Linux and macOS is for newlines to be indicated by
\n, the standard under Windows and its DOS predecessor is to use
instead \r\n. Here \n is the common escape sequence for the ASCII
line-feed character which has hex value 0A, \r is the common escape
sequence for the ASCII carriage-return character which has hex value
0D.

Some submissions were less flexible and required alternate input file
formatting. For example, some required no white-space around each
comma, some required the input to be a single comma-separated line of
100 numbers, and some only worked if the final line of the input file
did not have any line termination characters. A standard convention
is that the last line of a text file always ends with a line

termination character or character pair. Many programs reading text
files expect this, though some are also tolerant of these final-line
termination characters missing.

To test submissions and have them run properly, we ended up having to
use 15 different input formats, varying the number separators and the
existence and nature of the newline line terminators.

- space-comma-space separators
- comma-space separators
- comma separators
- DOS newlines, space-comma-space separators
- DOS newlines, comma-space separators
- DOS newlines, comma separators
- one line, space-comma-space separators
- one line, comma-space separators
- one line, comma separators
- no final newline, space-comma-space separators
- no final newline, comma-space separators
- no final newline, comma separators
- one line, no final newline, space-comma-space separators
- one line, no final newline, comma-space separators
- one line, no final newline, comma separators

We achieved the best auto-marker score for your code using formatting with
space-comma-space separators.

8. Auto marking output files

Please find attached two output files generated by the auto marker:

- auto-report.txt
- auto-report.geojson

The first is the a description of how the auto-marker arrived at its
score. The second is a visualisation of which heatmap squares in your
output were correct

END OF REPORT