COMP1721 Object-Oriented Programming

Coursework 1

1 Introduction

Your task is to write a Java program that analyzes levels of atmospheric nitrogen dioxide (NO_2) in Leeds, with the eventual goal of determining whether air pollution in the city breaches EU rules.

For some context to this problem, please read the following newspaper article about pollution levels in London in January 2017:

`http://bit.ly/airlondon2017`

We have provided you with three real datasets for 2016, measuring average levels of NO_2 in µg per cubic metre at 15-minute intervals throughout the year, at three different locations in the city: The Corn Exchange, Kirkstall Road and Temple Newsam. These datasets were obtained from

`https://datamillnorth.org/dataset/ratified-air-quality---nitrogen-dioxide`

There are three different levels of solution: basic, full and advanced. The assignment is worth 20% of your overall module grade.

2 Preparation

It is important that you follow the instructions below precisely. We strongly recommend that you perform these steps in a Linux environment.

1. Download `cwkfiles.zip` from the VLE and **put it into the top level of your repository**—i.e., the directory containing the `cwk1`, `cwk2` and `exercises` subdirectories.
2. Open a terminal window at the top-level of your repository, then unpack the Zip archive using the command `unzip cwkfiles.zip`. This should create subdirectories immediately below `cwk1` named `data`, `lib`, `src` and `test`. **Make sure that this is exactly what you see!** For example, you should *not* have a subdirectory of `cwk1` that is itself named `cwk1`.

   All of your code should be written in files in the `src` subdirectory. The other subdirectories should be left untouched. Be aware that you may lose marks if you alter anything in `data`, `lib` or `test`, or if you have the wrong directory structure to start with.
3. Remove `cwkfiles.zip` and use Git to add and commit the new files, then push your commit up to gitlab.com. The following commands, executed in the terminal at the top level of your repository, will achieve all of this:

```
git add cwk1
git commit -m "Initial files for Coursework 1"
git push origin master
```

3 Basic Solution

This is worth 17 marks.

For the basic solution, you must implement a class named `Measurement`, representing a measurement of NO_2 level at a single point in time, and a class named `PollutionDataset`, representing a sequence of such measurements. You must also implement a small program that uses these classes.

The two classes, their methods and their relationship are summarised by the UML diagram in Figure 1. Skeletal implementations of these classes are provided for you in files in the `src` directory. You must implement the missing code, guided by the 'To Do' comments and by the information in Section 3.1 below. The recommended procedure for implementation is outlined in Section 3.2.

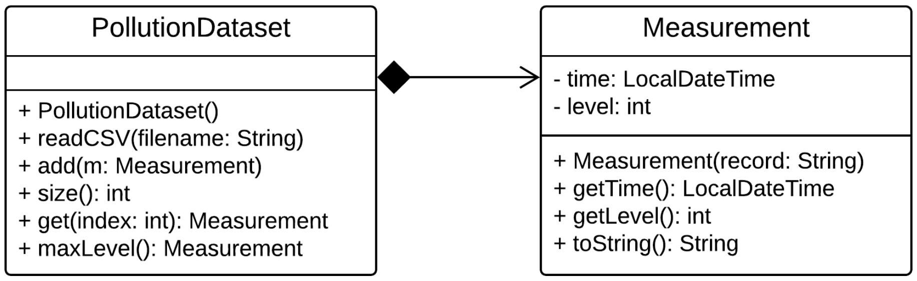

Figure 1: Classes required in the basic solution.

3.1 Class Implementation Notes

For the `Measurement` class:

- Time of measurement must be represented using the `LocalDateTime` class, from the `java.time` package. See the Java 8 API documentation for further information on this class.
- The `Measurement` constructor should initialise the `time` and `level` fields using the given string, representing a single record from the CSV file of pollution data. Here's an example:

  ```
  01/01/2016,00:15,84
  ```

  The three comma-separated elements of this record are date, time and level of NO_2 (in µg / m^3).

  The date and time values extracted from a record need to be combined into a single string. You can then parse this string using the `parse` method of the `DateTimeFormatter` object already defined for you in the `Measurement` class. (See the API documentation for further information.)

  Some records contain no value for NO_2 level. In these cases, your constructor should parse date and time as normal but should assign a value of `-1` to the `level` field, signifying 'no data'.
- If a record can't be parsed properly because it has too few elements or too many elements, your constructor should throw an instance of `DataException`. This class is already provided for you, in the `src` directory.
- The `toString` method should return a string that looks like this:

  ```
  2016-11-28T18:45, 174 µg/m³
  ```

  The first part of this is just the `time` field, rendered as a string. The 'micro' symbol in the units for NO_2 level can be produced using the Unicode escape sequence `\u00b5`. The 'cubed' superscript can be produced using `\u00b3`.

  In cases where `level` is `-1`, indicating that no proper value is available, the `toString` method should represent this using the words 'no data'.

For the `PollutionDataset` class:

- The default constructor should create a `PollutionDataset` that is empty but capable of storing `Measurement` objects.
- The `readCSV` method should read a CSV file containing air pollution data, create `Measurement` objects from the records in the file and then store these objects. Note that these new objects should *replace* any measurements that already exist in the dataset.
- The `add` method should add a new measurement to end of the current sequence of measurements.
- The `size` method should return the number of stored measurements.
- The `get` method should retrieve a stored measurement, given its position in the sequence (zero-based). If there are no stored measurements, it should throw `DataException`.
- The `maxLevel` method should return the measurement that has the largest NO_2 level in the entire dataset. In cases where multiple measurements have this level of NO_2, the first of these measurements should be returned. If there are no stored measurements, it should throw `DataException`.

3.2 Class Implementation Procedure

We suggest that you implement your solution step-by-step in the following way, using an automated testing procedure that we have set up for you.

The tests can be run using a bash shell script. This should work on Linux systems. It should also work in the macOS Terminal and in Microsoft's Windows Subsystem for Linux (but this hasn't been checked). If you are developing in a native Windows environment, it is possible to run the tests another way, using IntelliJ. Instructions on how to do this will be provided separately.

1. Start by moving into the `test` subdirectory in a terminal window. Run a set of tests for the basic solution via the following command:

   ```
   ./basictests
   ```

   You should see that all 13 tests fail, because none of the required code has been implemented in the classes yet.

2. Implement the missing code in the `Measurement` class. Do this one method at a time, removing the 'To Do' comments as you go. After implementing each method, run the `basictests` script again. If your implementation is correct, you should see the number of passing tests increase.

3. Do the same for the `PollutionDataset` class. A correct and fully functional implementation of both classes should result in all 13 tests passing.

Note that we will use this same script when marking your implementation. The tests will be run automatically in GitLab after you submit your work. You will get 1 mark for each passing test.

3.3 Using The Classes

The final step of the basic solution is to write a small program that uses the two classes. This program should be contained in a class named `Pollution`, in a file named `Pollution.java`. It should accept the name of a CSV data file on the command line. It should process the data in this file and then output the number of records processed and the details of the record with the maximum level of NO_2. For example, running the program with

```
java Pollution ../data/kirkstall.csv
```

should yield output similar to this:

```
35136 records processed
Max: 2016-09-08T19:00, 153 µg/m³
```

Your program should be friendly to users. If no command line argument has been supplied, it should display a helpful usage message. It should also intercept exceptions so that tracebacks are not seen by users. (Some information on the error should still be printed, however.)

A program that meets all of these requirements will earn you a further 4 marks.

4 Full Solution

This is worth an additional 11 marks.

The classes for the full solution are shown in Figure 2. The only difference between this and the basic solution is the addition of three new methods in `PollutionDataset`.

Some additions will also be needed to the program in `Pollution.java`.

4.1 Class Implementation Notes

- Each of the new methods should throw `DataException` if they are invoked on an empty dataset.
- The `minLevel` method should find and return the measurement having the minimum level of NO_2. Measurements with no meaningful value should be ignored.

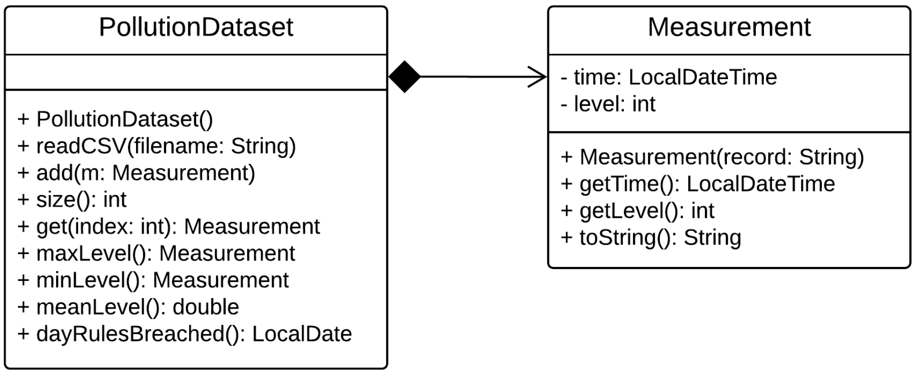
PollutionDataset
+ PollutionDataset()
+ readCSV(filename: String)
+ add(m: Measurement)
+ size(): int
+ get(index: int): Measurement
+ maxLevel(): Measurement
+ minLevel(): Measurement
+ meanLevel(): double
+ dayRulesBreached(): LocalDate
Measurement
- time: LocalDateTime
- level: int
+ Measurement(record: String)
+ getTime(): LocalDateTime
+ getLevel(): int
+ toString(): String

Figure 2: Classes required in the full solution.

- The `meanLevel` method should return the arithmetic mean of all NO_2 levels in the dataset, again ignoring measurements with no meaningful value for level.
- The `dayRulesBreached` method is a little more interesting and complicated than the others. It should return the day on which EU air pollution rules were breached at the location represented by the dataset. If EU rules have not been breached, it should return `null`.

  Details of these EU rules can be found in the newspaper article at `http://bit.ly/airlondon2017`. Note that the rules are based on *hourly* levels of NO_2.

4.2 Class Implementation Procedure

You can use a similar step-by-step process to that described in Section 3.2. You can run the `fulltests` script in the `test` directory to check your implementations of the three methods.

Note that `fulltests` expects all three methods to exist, and the tests won't compile unless this is the case. So the recommended first step is to implement them as 'dummy methods' or **stubs**—rather like the ones that were provided for you in the basic solution.

When you've implemented these stubs, you will be able to run `fulltests`, but the tests should all fail. Correct and fully functional implementations of the three methods should result in all 7 tests passing.

As in the basic solution, we will use this script when marking your implementation. The tests will be run automatically in GitLab after you submit your work. You will get 1 mark for each passing test.

4.3 Changes to Program

Modify the program in `Pollution.java` so that it displays

- Details of the measurement with the minimum level of NO_2, using the same format as for the maximum
- The mean level of NO_2 across the whole dataset
- A message indicating whether EU rules were breached and, if so, the date on which this occurred

A program that meets all of these requirements will earn you a further 4 marks.

5 Advanced Solution

NOTE: Attempt this additional work only if you are super-keen and finish the other parts quickly. It is worth only 4 marks and will require significant research into the JavaFX APIs.

1. Investigate JavaFX by reading Chapter 14 of the Liang book and trying out some of its examples, or by studying the documentation at

   `https://docs.oracle.com/javase/8/javase-clienttechnologies.htm`

   Pay particular attention to the section in the documentation on drawing charts.

2. Write a program called `PollutionPlot` that plots some of the data held in a `PollutionDataset` object. The type of plot is up to you. One possibility would be a bar chart showing average NO_2 level in each month of the year; another would be a line chart plotting hourly NO_2 totals for a user-selected period of time (e.g., the month of June). Figure 3 shows an example of the latter.

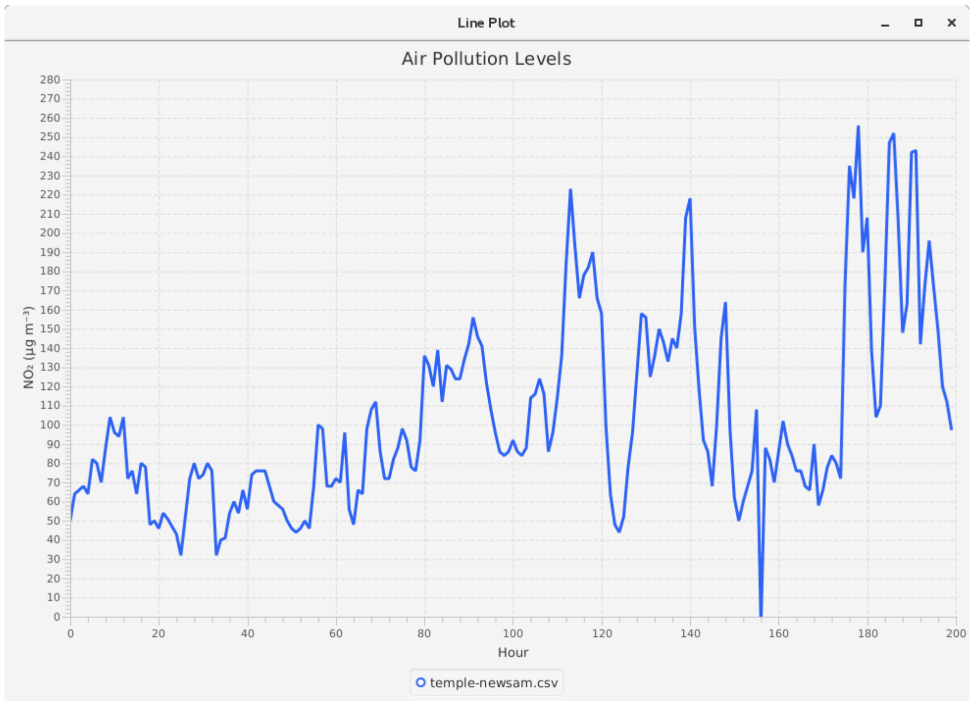

Figure 3: A plot of pollution data produced with JavaFX.

6 Submission

There is a README file in your project area on gitlab.com, explaining the submission process. Follow these instructions carefully. Remember to submit your Zip archive via the link provided in Minerva.

The deadline for submissions is **10 am on Thursday 1 March**. The standard university penalty of 5% of available marks per day will apply to late work, unless an extension has been arranged due to genuine extenuating circumstances.

Note that all submissions will be subject to automated plagiarism checking.

7 Marking

13	Tests for basic solution
4	Program for basic solution
7	Tests for full solution
4	Program for full solution
4	Program for advanced solution
6	Code quality (style, commenting, use of Java)
2	Use of Git and correct submission
40	