COMP1721 Object-Oriented Programming

Coursework 1: Creating & Using Classes

1 Introduction

Your task is to write a Java program that analyzes COVID-19 cases amongst staff, students and other individuals at the University of Leeds, as reported at

`https://coronavirus.leeds.ac.uk/statistics-and-support-available/`

You must do this by creating the classes described by the UML diagram in Figure 1. The class `CaseRecord` represents the cases recorded on a given date as a result of positive tests for COVID-19. The class `CovidDataset` represents a chronological sequence of these records.

There are three different levels of solution: basic, full and advanced. The UML diagram summarizes the requirements for the full solution. The advanced solution is aimed at people who complete the lower levels quickly and want some additional challenge.

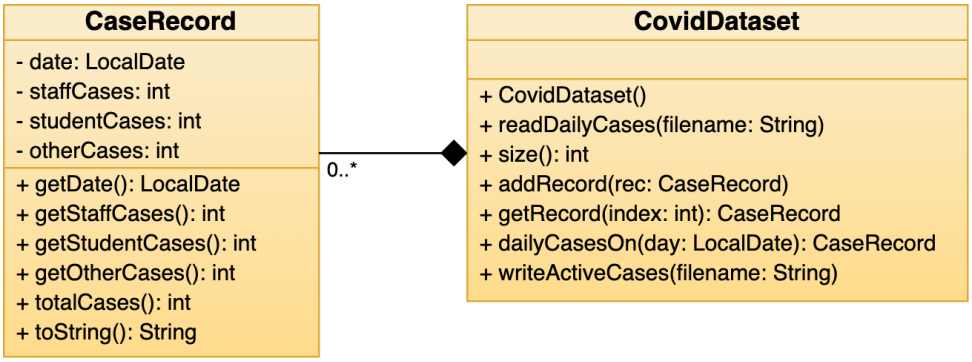

Figure 1: Main classes used in Coursework 1.

2 Preparation

It is important that you follow the instructions below *precisely*.

1. Download `cwk1-files.zip` from Minerva or Teams. **Put this file in the `coursework` directory of your repository**.
2. Unzip the Zip archive. You can do this from the command line in Linux, macOS and WSL 2 with `unzip cwk1-files.zip`.
3. Make sure that you have the following files and subdirectories immediately below `cwk1`:

   ```
   app/           config/      gradle/       README.html
   build.gradle   core/        gradlew       README.md
   chart/         datafiles/   gradlew.bat   settings.gradle
   ```

   IMPORTANT: Make sure that this is exactly what you see! For example, you should NOT have a subdirectory of `cwk1` that is itself named `cwk1`. Thus the path to the README file, relative to the repository directory, should be `coursework/cwk1/README.md`. Fix any problems with the directory structure before proceeding any further.
4. Remove `cwk1-files.zip`. Use Git to add and commit the new files, then push your commit up to gitlab.com. The following commands, executed in a terminal window while in the `coursework` directory of your repository, will achieve all of this:

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

3 Skeleton Classes

The first step is to create **skeletons** of the two classes shown in Figure 1. These are needed because we have provided a set of tests that we will use when marking your work, which you can also use yourself to assess your progress. These tests will be of no use until they compile, and they will only compile once you've provided **stubs** (dummy versions) of the methods in the two classes.

1. Edit `CaseRecord.java`, in the directory `core/src/main/java/comp1721/cwk1`. Within the class definition for `CaseRecord` add a constructor with four parameters:

   - A `LocalDate` object representing the date on which cases were recorded
   - An `int` value representing the number of staff cases
   - An `int` value representing the number of student cases
   - An `int` value representing the number of other cases

   Don't put anything in the body of the constructor yet. Note that you'll need to import `LocalDate` from the `java.time` package.

2. Implement the `CaseRecord` methods shown in Figure 1 as stubs. **Use the exact same method names and return types as shown in the UML diagram**. If a method is supposed to return a numeric value, make it return the value `0`. If a method is supposed to return an object of some kind, make it return `null`.

3. Edit `CovidDataset.java`, in the directory `core/src/main/java/comp1721/cwk1`. Within the class definition for `CovidDataset` add stubs for the methods shown in Figure 1. **Use the exact same method names, parameter types and return types as shown in the UML diagram**. If a method is supposed to return a numeric value, make it return the value `0`. If a method is supposed to return an object of some kind, make it return `null`. Constructors and methods that don't return anything should just have empty method bodies (i.e., nothing inside the braces).

4. Attempt to run the tests with

   ```
   ./gradlew :core:test
   ```

   (On Windows, omit the `./` from the start of this command.)

   If you've implemented the classes and their stub methods properly, the tests should compile and run but almost all of them should display FAILED in the terminal window.

   Consult the README file in `cwk1` for further details of how to run tests selectively and how to view test results in a web browser.

5. Commit the code changes to your Git repository.

4 Basic Solution

The features of the basic solution are worth **13 marks**. These marks are awarded for passing a set of tests. You can run these tests like this:

```
./gradlew :core:test --tests Basic
```

The `--tests Basic` in this command ensures that only the tests for the basic solution are run. You can omit it to run all the tests.

4.1 `CaseRecord`

Add the required fields to `CaseRecord`, then replace the method stubs with correct implementations. After implementing each method, rerun the tests, as shown above.

The getter methods should return the corresponding field values.

The `totalCases` method should return the total number of COVID cases recorded in a `CaseRecord`.

The `toString` method should return a string containing all the field values from the `CaseRecord`, formatted so that it looks like this example:

```
2020-10-07: 2 staff, 59 students, 0 other
```

An attempt to create a `CaseRecord` with a negative value for staff cases, student cases or other cases should result in a `DatasetException` being thrown, containing an appropriate error message. You have been provided with this exception class and do not need to write it yourself.

There are 7 tests relating specifically to the `CaseRecord` class in the Basic solution. Make sure all of these are passing before proceeding any further.

4.2 `CovidDataset`

This class represents an ordered sequence of `CaseRecord` objects. It will need a field capable of storing these objects, so the first step is to add this field, plus a default constructor that initializes it.

Next, turn your attention to the `addRecord`, `getRecord` and `size` method stubs. Replace each of these with the required implementation and rerun the tests to check whether you've done this correctly.

The `addRecord` method should append the given `CaseRecord` object to the end of the current sequence of records stored in a `CovidDataset`.

The `getRecord` method should return the `CaseRecord` object stored at the given position, specified as a zero-based integer index. It should throw a `DatasetException` if the supplied index is not valid.

The `size` method should return the number of `CaseRecord` objects stored in a `CovidDataset`.

Finally, implement the `dailyCasesOn` method. This method should find and return the `CaseRecord` object corresponding to the given date. If no `CaseRecord` can be found for the given date, the method should throw a `DatasetException`, containing an appropriate error message.

There are 6 tests relating specifically to the `CovidDatset` class in the basic solution. Make sure all of these are passing before proceeding any further.

5 Full Solution

The features of the full solution are worth a further **11 marks**. 6 marks are awarded for passing the automated tests and 5 marks are awarded for implementing a program that uses the two classes. You can run only the tests for the Full solution like this:

```
./gradlew :core:test --tests Full
```

Omit `--tests Full` from this command if you want to run all the tests.

5.1 `CovidDataset`

1. Consult the README in the `datafiles` directory, then examine the file `2020-daily.csv` in a text editor (not in a spreadsheet application). Each line of this CSV file (after the initial column headings) represents one record of the dataset. Your code will need to convert each of these lines into a suitable `CaseRecord` object, which should then be stored in the `CovidDataset` for later use.
2. Replace the stub for `readDailyCases` with an implementation that reads from a CSV file whose name is given by the method parameter. Note the following points regarding `readDailyCases`:
   - It should not catch any exceptions that occur during reading of a CSV file.
   - It will need an exception specification—either for `FileNotFoundException` or `IOException`, depending on the approach you've used to read the data.
   - Reading data from a file should clear out any data previously held in the `CovidDataset`.
   - It is OK for a dataset to be empty—i.e., it is OK for the CSV file to contain column headings but no case records.
3. Replace the stub for `writeActiveCases` with an implementation that computes the number of **active cases** on each day, for each category of individual (staff, student, other). The method should write the active case data out to a CSV file whose name is given by the method's filename parameter. The format for the output file should identical to the format of the daily cases file—i.e., same columns and column headings.

   The number of active cases on any given day is defined as the sum of the number of cases recorded over the past 10 days, including the given day.

Note the following points regarding `writeActiveCases`:

- You can assume that a `CovidDataset` holds consecutive days of data, with no missing days.
- You should skip the first nine days of data in a `CovidDataset`, since the tenth record in the dataset is the first for which a valid count of active cases can be produced.
- Your method should throw a `DatasetException` if there are not at least ten `CaseRecord` objects stored in the `CovidDataset`.
- Your method should not catch any exceptions that occur during writing of the CSV file.
- Your method will need an exception specification for `IOException`.

Make sure that all 6 tests pass before proceeding to the final part of the Full solution.

5.2 `ActiveCases` Program

Edit the file `ActiveCases.java`, in `app/src/main/java/comp1721/cwk1`. In this file, write a program that reads daily case data from a CSV file and then writes active cases to a different CSV file.

The names for these two files should be provided on the command line, with the daily cases file specified as the first command line argument, and the active cases file specified as the second command line argument. If the user of the program fails to provide two command line arguments, your program should print a helpful usage message and then terminate.

After reading and writing data, your program should display the number of records that were in the `CovidDataset`. It should display no other output (aside from error messages—see below).

Your program should catch any exceptions that occur during the reading, processing or writing of data. The error message associated with the caught exception should be displayed, then the program should terminate with a non-zero status code (to signify that an error of some kind has occurred).

You can run the program from the command line with

```
./gradlew :app:run
```

This will use files `datafiles/2020-daily.csv` and `datafiles/2020-active.csv` as the command line arguments. You can check the contents of the latter file to see if your program is behaving correctly.

6 Advanced Task

This task is more challenging and will require you to do some additional research. Also, it is worth only 5 marks. You should attempt it only if you manage to complete the Basic and Full solutions fairly quickly and easily.

1. Investigate **JavaFX**—e.g, by reading Chapter 6 of Eck's *Introduction to Programming Using Java* and trying out some of the examples.
2. Visit `http://bit.ly/jfxcharts` to learn about drawing charts in JavaFX.
3. Edit the file `CovidChart.java`, in `chart/src/main/java/comp1721/cwk1`. In this file, create a program that reads COVID case data from a CSV file and then draws a line chart showing how the total number of active cases changes over time.

   Your program should make use of the classes developed for the Basic and Full solutions. The x axis of the chart should be day of the year, which can be obtained from the `LocalDate` object associated with a `CaseRecord`.

   You should be able to compile and run the program with

   ```
   ./gradlew :chart:run
   ```

   This will be slow the first time it runs, as it needs to download additional dependencies.

   Figure 2 is an example of what this chart could look like. Your solution should show the data correctly but doesn't need to be identical to this example.

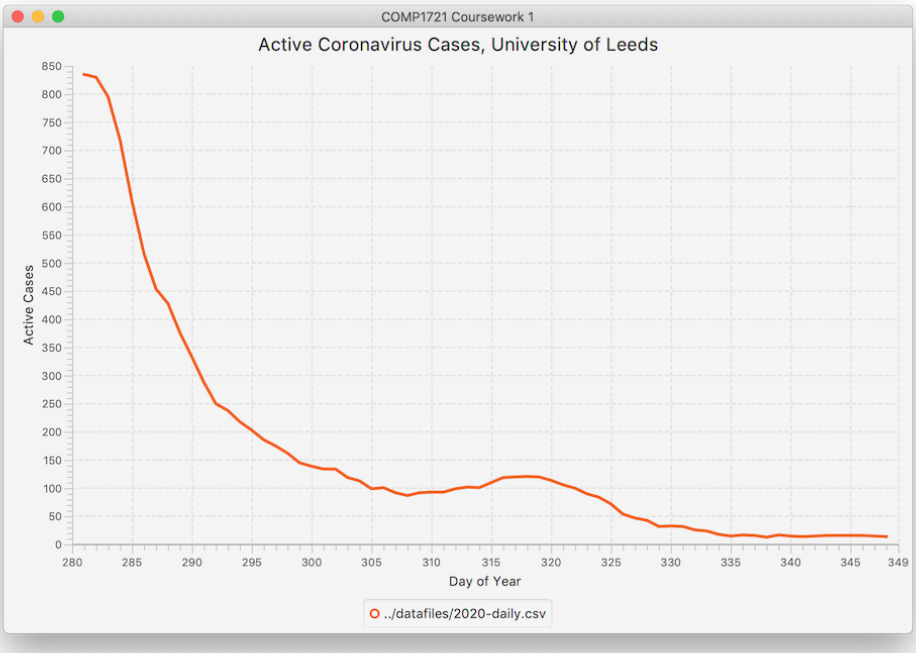

Figure 2: Example of a JavaFX chart for the Advanced task.

7 Submission

The final section of the README file for this coursework explains the submission process in detail. Please follow the submission instructions carefully.

The submission process will generate a file named `cwk1.zip` containing your solution. Please remember to submit this file to Minerva, via the link provided in the *Submit My Work* section.

The deadline for submissions is **10 am on Friday 5 March**.

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

8 Marking

13	Tests for basic solution
6	Tests for full solution
5	`ActiveCases` program for full solution
5	Advanced task (`CovidChart` program)
4	Coding style and comments
2	Use of version control
35	