4CCS1PPA Programming Practice and Applications

Coursework 4: London COVID-19 Statistics

Ishab Ahmed (K22009721), Harshraj Patel (K22018200), Saihan Marshall (K22011438) and Muhammad Ahsan Mahfuz (K22044399)
Team 3b1w

Base Tasks

All GUI elements were created in SceneBuilder and loaded into the controller classes using FXMLLoader. This was to save time by quickly creating the GUI layouts without writing and debugging code, to increase maintainability as separating the user interface from the code makes it easier to update the layout or design, and to improve collaboration within the team through the ease of sharing and modifying the FXML files.

Application Window - Implemented by Ishab Ahmed and Harshraj Patel

The application window (MainWindow.fxml controlled by MainWindowController) behaves as described in the task sheet.

The ability to move left and right through panels contained in the centre of the display has been implemented. The MainWindow is a BorderPane with the date pickers being set at the top, and the panel switching buttons being set at the bottom. The centre of the BorderPane is used for the panels.

The MainWindowController loads the FXML files of all of the panels, and stores their controllers in an array. When, for example, the right button is pressed, a field (*controllerIndex*) is incremented to get the controller of the next panel to contain in the centre of the MainWindow, it gets the view from the controller and places it in the centre of the BorderPane. An animation is also played when switching from one panel to another to provide visual feedback to the user during the transition instead of an abrupt switch.

Instead of using a drop-down box to select dates, there are two date picker objects. The date pickers give a more intutive calendar-style interface for the user to use. The MainWindowController calculates the farthest date and the newest date in the dataset and restrict the date pickers to be between this range. A label is also given to the user of this date range on the welcome screen. This prevents the user from picking a date that is not in the dataset, and having to learn what the range of the dataset is.

Loading and storing of data

A singleton design pattern is used to access the data. A singleton *Dataset* class has been created, so that there is only one instance of the dataset, which all controllers share, with a global point of access to that instance of the dataset. This ensures that all parts of the application use the same instance of the dataset. Moreover, it means only one instance of the relatively large list of CovidData objects is in memory at any given time, reducing memory usage and improving performance.

Using a singleton also removes the overhead of creating multiple instances of a class which processes the data, as was previously implemented. It provides better encapsulation as it hides the details of the data access from other parts of the application. With dependency injection, the *Dataset* would be explicitly passed, leading to more coupling between the code. Furthermore, as *Dataset* is accessed from many places in the code, passing around an instance of the class would lead to a significant amount of boilerplate code and make the source code difficult to understand, hence why a singleton is used.

The *getInstance()* method in *Dataset* has the *sychronized* keyword to ensure that only one instance of the *Dataset* is created in the case of a multi-threaded environment. Although BlueJ uses a single JavaFX thread, this prevents any issues if the code is taken out into a multi-threaded environment.

Panel 1: Welcome - Implemented by Ishab Ahmed

When the application is first opened, the user is greeted by a welcome, instructions on how to use the date pickers, as well as the range of dates that the dataset covers. The panel switching buttons are disabled until the user picks a valid date range.

If an invalid date range is chosen (e.g. *from* date is after *to* date), a label appears informing the user of the error, as well as an empty table to indicate that there is no data in the range that the user has chosen.

When a valid date range is chosen, a table appears showing the key information (only UK government data, not the data from Google's Mobility Report) for the period. This gives a very broad overview of the data that the user will see in finer, and more graphical detail in the following panels.

Panel 2: The Map - Implemented by Harshraj Patel

The second panel displays a map. The boroughs are represented by using JavaFX polygons. This allowed the creation of a geographically accurate map of London, with each polygon being the actual shape of the borough.

The map that gives a visual representation of the Covid death rates in the date range. It does this by summing up the new deaths for each record to get the deaths for the borough in the time period selected. The number of deaths in the borough in the date range is used to colour its respective polygon. Green represents a low number of deaths and red represents a high number of deaths, with a gradient between the green and red for values in between. The colour is determined by getting the number of deaths in the borough for the time period and getting it as a percentage of the highest deaths in the period. For example, if the highest deaths in the period was Harrow with 250 deaths, Harrow will be coloured with the deepest red and all other boroughs are coloured relative to this number (e.g. Camden with 120 deaths will be 120/250=48% between pure green and pure red).

When you hover over a borough in the map, a small box shows up showing the deaths in that period, as well as the percentage it is of the highest value in the date range. This allows the user to get the exact value that the colour is representing,

Borough Data

Upon a user selecting a date range, and after the appropriate visualisation of the data is displayed, it is possible for a user to click on any of the boroughs in order to see all of the Covid information for that specific borough. A new window pops up with a TableView displaying all of the data for that borough in the date range. At the top of the window, there is a drop-down menu that allows the user to sort by any one of the table columns, and whether they want the data in ascending or descending order.

Panel 3: Statistics - Implemented by Saihan Marshall

The statistics panel shows a series of statistics for the date range specified.

The panel first shows two statistics, presenting *(1)* the average retail and recreation mobility data, and *(2)* the average grocery and pharmacy mobility data. This is achieved by taking the mean of the data by summing up the all of the specified mobility data, and dividing by the number of data items. As there are many null items in the Google Mobility Data, the divisor is not the number of dates in the date range, but instead the number of dates with non-null as to not skew the average lower with null results.

As Jeffrey stated that "there's no right or wrong answer as long as you don't break your application", we interpreted null values to be N/A data rather than 0 as the Google Mobility Data is relative to a baseline, which is 0. With 0 being a valid value, we interpreted nulls as invalid values that should not be calculated with. For this reason, we also fixed what Jeffrey described as a "bug is in the base code" by making the helper functions in *CovidDataLoader* return null instead of -1 for empty data, as -1 is a valid data item (1% below the baseline of Google Mobility Data), therefore it makes more sense to interpret this as a null rather than assigning it -1.

The next statistic calculated is *(3)* total number of total deaths. Jeffrey defined this as "Total Deaths for all boroughs at the End date of the period". The total number of total deaths statistic was, therefore, calculated by summing up *Total Death* column for all boroughs on the last date in the date range (the *to* date).

The next statistic shown is *(4)* average of total cases. Jeffrey defined this as "Total Cases for all boroughs at the End date of the period / (number of boroughs)". The statistic was calculated by summing up the *Total Cases* column for all boroughs on the last date in the date range (the *to* date) and dividing by the number of boroughs with non-null values in this column.

The final statistic to be seen is *(5)* the date with the highest total deaths. Jeffrey defined this as "Always the End date of the selection period". For this reason, the final statistic is always the last date in the date range (the *to* date), as that is the date with the highest cumulative deaths (as it is the most recent date).

This makes *five* statistics implemented as required by the task sheet.

Challenge Tasks

Panel 4: Graphs - Implemented by Muhammad Ahsan Mahfuz

For the challenge task, the fourth panel provides a graph for the user to visually see how the data changed over time for each borough and for each field in the Covid dataset.

The user can select which borough they want to see data for, and for which field (e.g. Total Cases, New Deaths, Grocery and Pharmacy Mobility, etc.) in the dataset they want data graphed. A tooltip has also been added to each point that is graphed so that the user is able to see the exact data that the point is representing.

If you simply just graphed the data without any changes to the y-axis, there would be a large gap on the y-axis between zero and the minimum number in the data to be graphed (e.g. if the Total Cases for Hounslow is between 34350 and 95230, there would be a large gap between 0 and 34350). To solve this problem, the y-axis displays the data betwen the minimum and maximum value with a 10% padding around to make it more visually appealing for the user.

Unit Testing - Implemented by Ishab Ahmed

The *Dataset* class was tested due to being systemically important to our software. Without the *Dataset* class working as intended, all other classes will be unable to access and process the data to be displayed on the panels.

DatasetTest has nine test methods for the nine methods in *Dataset*. Each method of *Dataset* is thoroughly tested with a range of valid data, invalid data and edge cases where possible to test the methods as well as possible.

1. *testGetInstance()*: This method tests whether getInstance() returns a non-null instance of the Dataset class. First, the method checks if the instance retrieved for the first time is non-null. Then, it checks if retrieving the instance for a second time returns the same object.

2. *testGetData()*: This method tests whether getData() returns the expected list of CovidData objects from the dataset. First, the method ensures that the data has been loaded. Then, it checks that all elements in the array are of type CovidData.
3. *testGetDataInDateRange()*: This method tests whether the CovidData objects returned by getDataInRange() fall within the specified date range.
4. *testGetBoroughData()*: This method tests whether the CovidData objects returned by getBoroughData() falls within the given date range and belong to the specified borough. First, the method checks that each object's date falls within the specified range. Then, it checks that borough name matches the specified borough.
5. *testGetMostRecentDataWithFilter()*: This method tests that getMostRecentDataWithFilter() returns only the most recent records that match the given filter. First, the method gets the most recent data with the getTotalCases filter applied and checks that the data returned matches the filter and is the most recent available. Then, the method does the same, but with the getTotalDeaths filter applied instead.
6. *testGetBoroughs()*: This method tests whether getBoroughs() is returning the expected borough names or not. The method checks that each borough name in the returned array matches the expected borough names.
7. *testIsDateInRange()*: This method tests whether isDateInRange() returns true when the input date is within the date range (inclusive), and false when it's outside the range. The method checks the method's result for valid dates, invalid dates, and edge case dates.
8. testIsDateRangeValid(): This method tests whether isDateRangeValid() returns false when either date is null or the 'from' date is after the 'to' date, and true otherwise. The method checks the method's result for valid and invalid inputs.
9. testGetAverage(): This method tests whether the getAverage() method calculates the correct mean average, even when nulls are passed in. The method tests the method for various scenarios, including all null values, mixed values, and non-null values.