4CCS1PPA
William Costales k23081539 , Ye Win k23086194, Rojus Cesonic k23046252, Ruije Li k22029387

COURSEWORK 4
London Covid-19 Statistics GUI

Application Window (Ye):

Application Window serves as the main interface of the GUI application, encapsulating each different page within the window, allowing navigation.

- The forward and backwards buttons created in SceneBuilder are linked to methods *goForward() and goBackwards()* respectively. They enable users to navigate between different pages or sections within the application.
- Our methods *loadFile(), findFXMLFile()* are implemented to manage the loading of FXML files corresponding to different pages of the application. Using conditional statements, it iterates through its folder's files, picking only those ending with .FXML and containing an integer which identifies the order in which the pages should be displayed.
- The separation of classes for the main layout and the controller enhances the code cohesion, and maintainability and reduces coupling by isolating the GUI-relation functionality from the application logic.

Welcome Page (William):

Welcome Page serves as the initial interface displayed to the user upon opening the application. It provides information and instructions on how to run the application properly as well as displays the date selected.

- This page is designed using SceneBuilder, with a label created inside the window which links to our *WelcomePageController* class.
- We incorporate functionality to dynamically display the selected date range. This is achieved by adding listeners to the Start Date and End Date values provided by the

  DataManipulator singleton instance; whenever these dates are picked using the DatePicker, this method is invoked to update the label to show the dates.
- Within this *updateDateRangeLabel()* method, we incorporate conditional statements in order to validate the range of dates picked.

Map Page (Rojus, Ye):

The Map Page is a pivotal section of the GUI application, offering a visual representation and interactive exploration of COVID-19 data across each individual borough. The class MapController

- Upon initialisation, our MapController class populates a HashMap *boroughDictionary*, which maps the borough code and their names.
- With the *initialize()* method invoked, COVID-19 data is retrieved from the dataManipulator class, which is filtered based on the selected date range using conditional statements.
- Performance is optimised as *MapController* utilises methods belonging to the *dataManipulator class,* this reduces unnecessary iterations through the entire dataset, increasing the computational efficiency.
- Each individual borough of London is labelled with their corresponding name found in the *boroughDictionary.* As well as this, each button is dynamically determined based on the total number of deaths within each respective borough through conditional *if, else,* < statements. Methods *updateButtonColour() and determineButtonColour()* are responsible for this. Colour coding provides users with a visual indication of the severity of each borough.
- Users interact with the map by clicking individual borough buttons, *openNewWindow()* is called through an ActionEvent, which displays the data for the selected borough in a pop-up window.
- Within the selected borough window the user can sort the data via any of the columns within the table. This was implemented in the *getSortedData()* method which uses comparators within switch cases to sort the data via a certain attribute.
- To handle errors within this class, *try-catch* blocks are implemented such that interactions and data retrieval do not break our application.
- *updateRecordsInRange()* returns an array list of date-ranged records. This is accomplished by implementing an enhanced for loop to iterate through the records, only adding to the array list the records that fall between the date chosen.

Statistics Page (William, Ruije)

The *StatsPanelController* class serves as the controller for managing statistical information displayed in the stats panel of the application. It provides users with insights to various statistics to do with the time-range chosen.

- In SceneBuilder we created two labels; *infoLabel* and *statLabel*. The method *updateInfoLabel()* updates the labels with the appropriate statistical information based on the current indices (which correlates to a page inside the stat page).
- Navigation between each statistic is facilitated with methods *forwardStat()* and *prevStat()*.
- Statistical calculations include the total deaths across all records, average of total cases, average park GMD data and average transit GMD data. Each of these statistics are calculated in respective methods, through using enhanced for loops to iterate through the records, retrieving relevant data.

Graphical Representation Page Challenge Task (Rojus)

The *ComparatorController* class associated with our challenge task manages the display and sorting of the data for specific boroughs of the GUI application, storing it as a bar graph. It allows functionality to compare two borough's on their statistics.

- Upon initialisation, the ComboBoxes are populated with a list of borough names as well as data items for selection.
- To update the chart created in SceneBuilder, the methods *getBarChartData()* and *updateBarChart()* are used to retrieve selected data from the combo boxes for selected boroughs, and then update the BarChart.
- To handle the event of choosing a borough from the combo boxes, we created a method *handleBoroughSelection()* which ensures selected boroughs are unique by checking with conditional statements. It also ensures the bar chart is cleared before setting the new data so that upon any changes the new data is instantly displayed.
- Sorting functionality is added through the use of the ComboBox as well as the methods *handleSorterSelection(), getSortedData()*. The use of switch cases allows sorting through different data specifications (new cases, new deaths). Note, new deaths data has been scaled by a factor of 100 so it's easier to visualise for the user on the bar chart.

Unit Testing (Ye)

The DataManipulatorTest class contains a suite of tests to ensure the correctness and reliability of the DataManipulator Class methods. The reason for choosing to unit test this class out of all the classes is because it is the main class that contains complex methods to handle all the loading, manipulation, processing, validation and verification of the CovidData needed that the entire program depends on.

- *testGetInstance()* method verifies that *getInstance()* method in DataManipulator returns the same instance if the class to ensure it remains a singleton class
- *testCheckValidDate()* method verifies that *checkValidDate()* method in DataManipulator correctly validates the chosen dates against the valid dates in the stored CovidData records
- *testGetFilterByBorough()* method verifies that *getFilterByBorough()* method in DataManipulator correctly filters CovidData records by the selected borough name
- *testGetNewestTotalDeath()* method verifies that *getNewestTotalDeath()* method in DataManipulator correctly retrieves the total death of the latest CovidData record.
- *testUpdateRecordsInRange()* method verifies that *updateRecordsInRange()* method in DataManipulator correctly updates the list of records based on the specified specified data range