INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

Informatics Institute of Technology

Department of computing

(B.Eng.) in Software Engineering

Module: 4COSC010C Programming Principles 02

Module Leader: Mr. Guhanathan Poravi

Report

Student ID	:	2019163
Student UoW ID	:	w1761196
Student First Name	:	Mohamed
Student Surname	:	Raneez

Main Class – TrainStation

```
package sample;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.stage.Stage;

import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDate;
import java.util.*;

import com.mongodb.MongoClientSettings;
import com.mongodb.client.*;
import org.bson.Document;
import org.bson.codecs.configuration.CodecRegistry;
import org.bson.codecs.pojo.PojoCodecProvider;
import static com.mongodb.client.model.Projections.exclude;
import static org.bson.codecs.configuration.CodecRegistries.fromProviders;
import static org.bson.codecs.configuration.CodecRegistries.fromRegistries;

public class TrainStation extends Application {
    //objects from the gui and queue classes
    private GuiElements guiElements = new GuiElements();
    private PassengerQueue trainQueue = new PassengerQueue();

    //variables to be saved
    private List<Passenger> waitingRoom = new ArrayList<>();
    private List<Passenger> onTrain = new ArrayList<>();
    private int firstPersonIndex, newPassengerIndexForReport;       //two
index values, first - for the view method, next for the report index
    private int queueLength = trainQueue.getQueueArray().length;
    //this variable is required for the view display of the queue, cuz it
changes when you delete

    //three scenes for view method
    private Scene sceneView, sceneView1, sceneView2;

    @Override
    public void start(Stage primaryStage){//icon for the window
        primaryStage.getIcons().add(new
Image("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/Pictures/icon.png")
);

        for (int i=0;i<42;i++){
            onTrain.add(new Passenger());                   //filling the data
structures with 42 passenger objects to prevent null pointer exceptions
            trainQueue.getQueueArray()[i] = new Passenger();
            waitingRoom.add(new Passenger());
        }

        selectDate(primaryStage);                           //calling the date
gui which runs the entire program
    }
```

```
    ////////////////////////////////////////////////////////*****DATE
SELECTION*****////////////////////////////////////////////////////////////
    private void selectDate(Stage window){
        ////////////////////////////////////ALL REQUIRED GUI ELEMENTS CALLED
BY THE OBJECT CREATED/////////////////////////////////////////////////////
        Label label = guiElements.labels(50, 0, "Denuwara Menike Intercity
Express Train", "label");
        Label label1 = guiElements.labels(140, 70, "-First Class AC
Compartment-", "label");
        ComboBox<String> allJourneys = guiElements.allJourneys();
        Button confirm = guiElements.buttons("CONFIRM", 220, 330,
"continueBtn"); confirm.setMinWidth(400);
        DatePicker datePicker = guiElements.datePicker();
        AnchorPane anchor = guiElements.anchor();

        confirm.setOnAction(event -> {
            window.close();                                 //closing stage,
getting the date and journey choices
            LocalDate localDate = datePicker.getValue();
            String journey = allJourneys.getValue();
            loadInitial(journey, localDate);                //calling the
method that loads data from the CW-01
            System.out.println("Now Starting Up");

            try {
                for (int i=0;i<3;i++) {
                    Thread.sleep(800);
                    System.out.print(".");
                }
            } catch (InterruptedException e) {
                System.out.println("Something went WRONG");
            }

            try {
                Thread.sleep(800);
                displayMenu(journey, localDate, window);    //call the menu
once the data has been loaded
            } catch (InterruptedException e) {
                System.out.println("Something went WRONG");
            }
        });

        ////////////////////////////////////////////////////////WINDOW CLOSE
REQUEST ////////////////////////////////////////////////////////////////
        window.setOnCloseRequest(event -> {
            event.consume();                    //tells the window that it is
taking responsibility on handling window closing
            guiElements.closeDateGui(window);   //closes date gui, called from
the elements class
        });
```

```
        ////////////////////////////////////////////////////////////SCENE
CREATION//////////////////////////////////////////////////////////////
        anchor.getChildren().addAll(label, label1, datePicker, confirm,
allJourneys);   //container to hold all gui
        Scene scene = guiElements.scene(anchor, 800, 400, "style.css");
        window.setScene(scene);                 //scene of the gui, method
called from the class, set title and show the window
        window.setTitle("QUEUE | DEPARTURE");
        window.show();
    }


    ////////////////////////////////////////////////////*****INITIAL
LOAD*****////////////////////////////////////////////////////////////
    private void loadInitial(String journey, LocalDate localDate){
        CodecRegistry pojoCodecRegistry =
fromRegistries(MongoClientSettings.getDefaultCodecRegistry(),

fromProviders(PojoCodecProvider.builder().automatic(true).build()));
        MongoClientSettings settings =
MongoClientSettings.builder().codecRegistry(pojoCodecRegistry).build();
        MongoClient mongoClient = MongoClients.create(settings);
        MongoDatabase database = mongoClient.getDatabase("TrainBooking");
//creating client and getting database
        MongoCollection<Passenger> collection =
database.getCollection(localDate + " " + journey + " reservation",
Passenger.class);
        FindIterable<Passenger> eachDocument = collection.find();
//getting the collection and the document inside
        eachDocument.projection(exclude("_id"));
//removing off id field so that there's only name and seat

        List<Passenger> temp = new ArrayList<>();
        for (Passenger doc : eachDocument) {
            temp.add(doc);
//adding the iterable document into a temporary document
        }

        //for loop to set the loaded passengers onto the waiting room
        for (int i=0;i<temp.size();i++){
            waitingRoom.set(i, temp.get(i));
        }
        ////////////////////////////////////////SORTING THE WAITING ROOM IN
ASCENDING ORDER OF SEAT NO.//////////////////////////////////////////////
        Passenger temp3;    //temp variable required for bubble sorting
        //the following bubble sort algorithm is to sort the waiting room
passengers in ascending order based on seat
        for (int i=0; i<waitingRoom.size()-1; i++){
            for (int j = 0; j<waitingRoom.size()-i-1; j++){
                if (waitingRoom.get(j).getSeatsBooked() != 0 &&
waitingRoom.get(j+1).getSeatsBooked() != 0) {
                    if (waitingRoom.get(j).getSeatsBooked() >
waitingRoom.get(j + 1).getSeatsBooked()) {
                        temp3 = waitingRoom.get(j);
                        waitingRoom.set(j, waitingRoom.get(j + 1));
                        waitingRoom.set(j + 1, temp3);
                    }
                }
            }
        }
```

```
        ////////////////////////////////////////////////////WAITING ROOM
CHECKER GUI//////////////////////////////////////////////////////////////
        Stage window = new Stage();
        window.getIcons().add(new
Image("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/Pictures/icon.png")
);
        window.setTitle("QUEUE | DEPARTURE");
        ImageView logo =
guiElements.imageViewLay("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/
Pictures/LOGO1.png",
                360, 0, 100, 200);
        Label journeyLabel = guiElements.labels(50, 10, "Journey: " + journey,
"jDLabels");
        Label dateLabel = guiElements.labels(690, 10, "Date:  " + localDate,
"jDLabels");
        Label labels = guiElements.labels(50, 120, "| \t\tPassenger Name\t\t|
\t\t NIC \t\t\t | \t\tSeat Booked\t\t |",
                "checkHeader");
        Button submitBtn = guiElements.buttons("Confirm", 250, 630,
"waitingOkBtn"); submitBtn.setMinWidth(400);
        AnchorPane anchor = guiElements.reportAnchor();
        VBox vbox = guiElements.vbox(15, 55, 180);
        ScrollPane scrollPane = guiElements.reportScroll(800, 430, 50, 180, "-
fx-background: #222;" +
                "fx-background-color: #222; -fx-border-color: #fff");
        ////////////////////////////////////////////////////////GUI
ELEMENTS//////////////////////////////////////////////////////////////////
//

        CheckBox[] checkBoxes = new CheckBox[42];           //array that holds
all the checkboxes

        //for loop to add the passengers loaded from CW-01 into the GUI
        for (int i=0;i<waitingRoom.size();i++) {
            if (waitingRoom.get(i).getName() != null){
                if (waitingRoom.get(i).getName().length()>=5 &&
waitingRoom.get(i).getName().length()<10) {
                    checkBoxes[i] = new CheckBox("\t\t" +
waitingRoom.get(i).getName() + "\t\t\t\t\t\t\t\t" +
waitingRoom.get(i).getNIC() +
                            "\t\t\t\t\t\t\t\t\t\t" + "   " +
waitingRoom.get(i).getSeatsBooked());
                }else if (waitingRoom.get(i).getName().length()>=10){
                    checkBoxes[i] = new CheckBox("\t\t" +
waitingRoom.get(i).getName() + "\t\t\t\t\t\t" + waitingRoom.get(i).getNIC()
+
                            "\t\t\t\t\t\t\t\t\t\t" + "   " +
waitingRoom.get(i).getSeatsBooked());
                }
                checkBoxes[i].setMinWidth(750);
                checkBoxes[i].setMaxWidth(750);
                checkBoxes[i].setId("checkWaiting");
                checkBoxes[i].setSelected(true);
                vbox.getChildren().add(checkBoxes[i]);
            }
        }
        scrollPane.setContent(vbox);                            //setting the
content into a scrollPane
```

```
        ////////////////////////////////////////////////////////WINDOW ON
CLOSE REQUEST//////////////////////////////////////////////////////////////
        submitBtn.setOnAction(event -> window.close());

        window.setOnCloseRequest(event ->{
            event.consume();
            window.close();
        });

        ////////////////////////////////////////////////////////SCENE
CREATION//////////////////////////////////////////////////////////////
        anchor.getChildren().addAll(scrollPane, labels, submitBtn, logo,
dateLabel, journeyLabel);
        Scene scene = guiElements.scene(anchor, 900, 690, "style2.css");
        window.setScene(scene);                                 //adding
everything into the main parent container, setting scene and showing
        window.showAndWait();                                   //we use show
and wait to pause the program execution temporarily

        ////////////////////////////////////////////////////////WAITING
ROOM UPDATER//////////////////////////////////////////////////////////////
        //for loop to check attendance of passengers, checks whether each
index of the checkbox array are selected, if not, change that index to a
        //new passenger, we cant delete the passenger since that will affect
the index values
        for (int i=0;i<checkBoxes.length;i++){
            if (checkBoxes[i] != null && !checkBoxes[i].isSelected()){
                waitingRoom.set(i, new Passenger());
            }
        }

        //in this section, we remove all the null passengers, and add them to
the end of the waiting room list, in the end it will still be of size 42
        int numberDeleted = 0;
        List<Passenger> tempWaiting = new ArrayList<>(waitingRoom);
        for (Passenger passenger : tempWaiting){
            if (passenger.getSeatsBooked() == 0){
                waitingRoom.remove(passenger);
                numberDeleted++;
            }
        }
        for (int i=0;i<numberDeleted;i++){
            waitingRoom.add(new Passenger());
        }

        try {
            for (int i=0;i<3;i++) {
                Thread.sleep(800);
                System.out.print(".");
            }
            Thread.sleep(1000);
            System.out.println("\nBooking Details Were Successfully
Loaded\n...");
        } catch (InterruptedException e) {
            System.out.println("something went WRONG");
        }
```

```
        //outputting passenger details of everyone in the waiting room
        for(Passenger passenger : waitingRoom){
            if (passenger.getName()!= null)
                System.out.println("Passenger Name: " + passenger.getName() +
" | Seat Booked: " + passenger.getSeatsBooked());
        }
    }


    ////////////////////////////////////////////////////////////*****MENU
OPTIONS*****////////////////////////////////////////////////////////////
    private void displayMenu(String journey, LocalDate localDate, Stage
window) throws InterruptedException {
        //creating scanner object and obtaining user choice, and converting to
lower case to facilitate checking
        Scanner sc = new Scanner(System.in);
        System.out.println("\nWelcome to the Denuwara Menike Intercity Express
Train. Please choose an option to continue");
        System.out.println("\"A\" - Add Passenger to Train Queue \n\"V\" -
View Train Queue \n\"D\" - Delete Passenger from Train Queue " +
                "\n\"S\" - Save data \n\"L\" - Load data" + "\n\"R\" - Run
Simulation" +  "\n\"Q\" - Quit");

        String answer = sc.next().toLowerCase();

        switch (answer){
            case "a":           //calling the appropriate methods depending on
users option choice
                addPassengers(journey, localDate, window);
                break;
            case "v":
                viewPassengers(journey, localDate, window);
                break;
            case "d":
                deletePassengers(journey, localDate, window);
                break;
            case "s":
                savePassengers(journey, localDate, window);
                break;
            case "l":
                loadPassengers(journey, localDate, window);
                break;
            case "r":
                runSimulation(journey, localDate, window);
                break;
            case "q":
                System.out.println("Thank you for choosing denuwara Menike
Intercity :)");
                System.exit(0);
                break;
            default:
                System.out.println("Please enter a VALID option");
                displayMenu(journey, localDate, window);
        }
    }
```

```
    ////////////////////////////////////////////////////////////*****VIEW
METHOD*****////////////////////////////////////////////////////////////
    private void viewPassengers(String journey, LocalDate localDate, Stage
window){
        ////////////////////////////////////////////////////**QUEUE
VISUALIZATION**////////////////////////////////////////////////////////////
        Label trainName = guiElements.labels(170, 380, "D E N U W A R A  M E N
I K E  I N T E R C I T Y", "trainName");
        Label trainQueueLabel = guiElements.labels(490, 630, "T R A I N  Q U E
U E", "trainQueueLabel");
        Label counter = guiElements.labels(10, 640, "COUNTER", "counter");
        Label toTrainArrow = guiElements.labels(120, 660, "⬇To Train",
"arrowHead");
        Button trainViewBtn = guiElements.buttons("Go To Train", 10, 10,
"trainViewBtn");
        Button waitingRoomBtn = guiElements.buttons("Go To Waiting Room", 200,
10, "trainViewBtn");
        waitingRoomBtn.setOnAction(event -> window.setScene(sceneView2));
        trainViewBtn.setOnAction(event -> window.setScene(sceneView1));//on
action, change scene
        ImageView imageViewLay =
guiElements.imageViewLay("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/
Pictures/TrainView2.png",
                50, 10, 370, 500);
        ImageView imageViewLay2 =
guiElements.imageViewLay("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/
Pictures/TrainView7.png",
                800, 10, 370, 500);
        HBox layoutSeats = guiElements.hbox(2, 10, 600);
        AnchorPane anchor = guiElements.anchor();
        ////////////////////////////////////////////////////ALL GUI
ELEMENTS////////////////////////////////////////////////////////////

        int lblIndex = 0;                                          //this
particular variable is only used to align the positions

        //for loop that creates all positions, queue length is the variable
that changes - starts off with 42 slots and with each delete the slots
        //decrease since we loop from the first person index, only passengers
from 1 after the passenger who currently boarded are shown; mean time
        //also showing forward movement as the slots decrease. this index i
required since we use the queue array here to set name and seats,
        //queue array by itself doesn't change unless Delete is called,
therefore in order to visualize movement we have this variable which
        //increments whenever we add a passenger onto the train, or rather
remove them from the queue, so basically it refers to the passenger after
        //the passenger who was added onto the train.
        for (int i = firstPersonIndex; i<queueLength; i++){
            Button passengerFirstSeat = new Button();
            Label nameLbl = guiElements.labels(6 + lblIndex, 560, "empty",
"nameLbl");
            passengerFirstSeat.setId("emptyQueue");
            passengerFirstSeat.setText("✕");
```

```
        if (trainQueue.getQueueArray()[i].getName() != null){       //if the
particular position holds a passenger
                passengerFirstSeat.setId("viewQueue");                  //set
their details instead of the word empty
                if (trainQueue.getQueueArray()[i].getName().length()>=5 &&
trainQueue.getQueueArray()[i].getName().length()<=8) {
                    nameLbl = guiElements.labels(lblIndex - 5, 550,
trainQueue.getQueueArray()[i].getName() + ", " +
                            trainQueue.getQueueArray()[i].getSeatsBooked(),
"nameLbl");
                }else if (trainQueue.getQueueArray()[i].getName().length()>8
&& trainQueue.getQueueArray()[i].getName().length()<10) {
                    nameLbl = guiElements.labels(lblIndex - 13, 540,
trainQueue.getQueueArray()[i].getName() + ", " +
                            trainQueue.getQueueArray()[i].getSeatsBooked(),
"nameLbl");
                }else if (trainQueue.getQueueArray()[i].getName().length()>10
&& trainQueue.getQueueArray()[i].getName().length()<13) {
                    nameLbl = guiElements.labels(lblIndex - 15, 540,
trainQueue.getQueueArray()[i].getName() + ", " +
                            trainQueue.getQueueArray()[i].getSeatsBooked(),
"nameLbl");
                }else{
                    nameLbl = guiElements.labels(lblIndex - 20, 540,
trainQueue.getQueueArray()[i].getName() + ", " +
                            trainQueue.getQueueArray()[i].getSeatsBooked(),
"nameLbl");
                }
            }
            anchor.getChildren().add(nameLbl);
//adding the label and button into the containers
            layoutSeats.getChildren().add(passengerFirstSeat);
            lblIndex+=32;
//increasing the index value for the next node
        }


        ////////////////////////////////////////////////////**TRAIN
VISUALIZATION**////////////////////////////////////////////////////////////
/
        Label trainName2 = guiElements.labels(190, 600, "T R A I N  O N - B O
A R D  D I S P L A Y", "trainName");
        Button queueViewBtn = guiElements.buttons("Go To Queue", 10, 10,
"trainViewBtn");
        Button waitingRoomBtn1 = guiElements.buttons("Go To Waiting Room",
200, 10, "trainViewBtn");
        waitingRoomBtn1.setOnAction(event -> window.setScene(sceneView2));
        queueViewBtn.setOnAction(event -> window.setScene(sceneView));
        ImageView imageViewLay4 =
guiElements.imageViewLay("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/
Pictures/TrainView8.png",
                20, 60, 370, 400);
        TilePane layoutTrain = guiElements.tilePane(430, 20);
        AnchorPane anchor1 = guiElements.anchor();

        Label[] labelSeats = new Label[42];
        Button[] allSeats = new Button[42];
```

```
        //loop through the 42 sized list on the passengers who have boarded,
and create all 42 seats, along with empty labels beside each
        for (int i=1;i<=onTrain.size();i++) {
            if (i <= 9) {
                allSeats[i-1] = new Button("0" + i);
            } else {
                allSeats[i-1] = new Button("" + i);
            }
            labelSeats[i-1] = new Label("empty", allSeats[i-1]);
            labelSeats[i-1].setMinWidth(150);
            labelSeats[i-1].setId("labelSeat");
            allSeats[i-1].setId("emptySeat");
            layoutTrain.getChildren().add(labelSeats[i-1]);
        }

        //loop through waiting room and set each passengers, that have
arrived, corresponding seats text to their names
        for (Passenger passenger : waitingRoom){
            if(passenger.getName() != null)
labelSeats[passenger.getSeatsBooked()-1].setText(passenger.getName());}

        //loop through queue and set each passengers corresponding seats text
to their names
        for (Passenger passenger : trainQueue.getUpdatedQueue()){
            if(passenger.getName() != null)
labelSeats[passenger.getSeatsBooked()-1].setText(passenger.getName());}
        //this for loop is also needed as the waiting room is updated and once
they are added into the queue their label will become empty

        //if they have entered the train too, change border color to indicate
"on-board" and add name label
        for (Passenger passenger : onTrain){
            if (passenger.getName() != null) {
                labelSeats[passenger.getSeatsBooked()-
1].setText(passenger.getName());
                allSeats[passenger.getSeatsBooked()-1].setStyle("-fx-border-
color: red");
            }
        }
        //the adding of names is also required for the same reason as the
above enhanced for loop


        ////////////////////////////////////////////////**WAITING ROOM
VISUALIZATION**////////////////////////////////////////////////////////////
        Label waitingRoomLbl = guiElements.labels(420, 10, "W A I T I N G  R O
O M", "trainName");
        Button queueViewBtn1 = guiElements.buttons("Go To Queue", 10, 10,
"trainViewBtn");
        Button trainViewBtn1 = guiElements.buttons("Go To Train", 200, 10,
"trainViewBtn");
        trainViewBtn1.setOnAction(event -> window.setScene(sceneView1));
        queueViewBtn1.setOnAction(event -> window.setScene(sceneView));
        TilePane layoutWaiting = guiElements.tilePane(200, 100);
        AnchorPane anchor2 = guiElements.anchor();

        Label[] labelSeats1 = new Label[42];
        Button[] allSeats1 = new Button[42];
```

```
        //for loop to create 42 seats in the waiting room
        for (int i=1;i<=onTrain.size();i++) {
            allSeats1[i-1] = new Button("❌");
            labelSeats1[i-1] = new Label("empty", allSeats1[i-1]);
            labelSeats1[i-1].setMinWidth(150);
            labelSeats1[i-1].setId("labelSeat");
            allSeats1[i-1].setId("emptySeat");
            layoutWaiting.getChildren().add(labelSeats1[i-1]);
        }

        //loop through the waiting room and add the names of the passengers
currently in the waiting room
        for (Passenger passenger : waitingRoom){
            if(passenger.getName() != null) {
                labelSeats1[passenger.getSeatsBooked()-
1].setText(passenger.getName());
                allSeats1[passenger.getSeatsBooked()-1].setStyle("-fx-border-
color: red");
            }
        }

        ////////////////////////////////////////////////////////WINDOW CLOSE
REQUEST//////////////////////////////////////////////////////////////
        window.setOnCloseRequest(event -> {
            event.consume();
            closeScenes(journey, localDate, window);
        });

        ////////////////////////////////////////////////////////////SCENE
CREATION//////////////////////////////////////////////////////////////
        //adding all elements to the corresponding containers
        anchor.getChildren().addAll(trainViewBtn, trainName, trainQueueLabel,
imageViewLay, imageViewLay2,layoutSeats, counter, toTrainArrow,
                waitingRoomBtn);
        anchor1.getChildren().addAll(layoutTrain, queueViewBtn, imageViewLay4,
trainName2, waitingRoomBtn1);
        anchor2.getChildren().addAll(queueViewBtn1, trainViewBtn1,
layoutWaiting, waitingRoomLbl);
        sceneView1 = guiElements.scene(anchor1, 1366, 705, "style2.css");
        sceneView = guiElements.scene(anchor, 1366, 705, "style2.css");
//connecting to the css files, setting initial scene
        sceneView2 = guiElements.scene(anchor2, 1366, 705, "style2.css");
//and showing
        window.setScene(sceneView);
        window.show();
    }


    ////////////////////////////////////////////////////////******ADD
METHOD******//////////////////////////////////////////////////////////////
    private void addPassengers(String journey, LocalDate localDate, Stage
window) throws InterruptedException {
        int randomCount = new Random().nextInt(6) + 1;  //random count, for
number of people to join queue, between 1 and 6
        boolean emptyFlag = false;
```

```
        //loops randomCount no. of times, gets the corresponding index of
waiting room and adds into the queue
        for (int i = 0; i <randomCount; i++) {
            if (!waitingRoom.isEmpty()) {                       //only add if
the waiting room isn't empty
                if (waitingRoom.size() > randomCount) {         //to prevent
adding of nulls, if the randomCount > waitingROom.size()
                    if (waitingRoom.get(i).getName() != null) {
                        trainQueue.add(waitingRoom.get(i));
                    }
                } else {                                        //this boolean
flag is to signify that the waiting room is empty, as once all the
                    emptyFlag = true;                           //passengers
join, the list will have only null values/ be empty
                }
            }
        }

        trainQueue.setUpdatedQueue();                           //the set
updated queue is now called to get the list of current queue passengers

        //loop through the current queue, and remove those passengers from the
waiting room
        for (Passenger passenger : trainQueue.getUpdatedQueue()){
            waitingRoom.remove(passenger);
        }

        //prints if waiting room is empty
        if (waitingRoom.isEmpty())
            System.out.println("Waiting room is empty");
         else if (waitingRoom.get(0).getName() == null)
             System.out.println("Waiting room is empty");

        //this particular section is to add the left over passengers, if the
generated randomCount was greater than the number of passengers in the room
        if (emptyFlag){
            for (Passenger passenger : waitingRoom) {
                if (passenger.getName() != null && !waitingRoom.isEmpty()) {
                    trainQueue.add(passenger);
                }
            }
            waitingRoom.clear();                                //clear the
waiting room if there are nulls, and print the empty message
            System.out.println("Waiting Room is Now Empty");
        }

        //call method to update the queue, again, required cuz if we added
passengers in the above condition it should get updated
        trainQueue.setUpdatedQueue();
```

```
        //if queue is full, this alert GUI opens up
        if (trainQueue.isFull()){
            Stage fullQueue = new Stage();
            ImageView logo =
guiElements.imageViewLay("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/
Pictures/LOGO1.png",
                    420, 0, 100, 200);
            Label journeyLabel = guiElements.labels(40, 20, "Journey: " +
journey, "jDLabels");
            Label dateLabel = guiElements.labels(800, 20, "Date:  " +
localDate, "jDLabels");
            Button okBtn = guiElements.buttons("CONTINUE", 410, 210,
"trainViewBtn"); okBtn.setMinWidth(200);
            HBox hbox = guiElements.hbox(10, 125, 100);
            Label fullLabel = guiElements.labels(125, 100, "⚠T R A I N  Q U
E U E  I S  C U R R E N T L Y  F U L L ⚠",
                    "fullQueue");
            hbox.getChildren().add(fullLabel);
            AnchorPane anchor = guiElements.anchor();
            ////////////////////////////////////////////////////////////GUI
ELEMENTS//////////////////////////////////////////////////////////////////

            ////////////////////////////////////////////////////////WINDOW ON
CLOSE REQUEST////////////////////////////////////////////////////////////
            okBtn.setOnAction(event -> fullQueue.close());

            fullQueue.setOnCloseRequest(event -> {
                event.consume();
                fullQueue.close();
            });

            ////////////////////////////////////////////////////////////SCENE
CREATION/////////////////////////////////////////////////////////////////
            fullQueue.getIcons().add(new
Image("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/Pictures/icon.png")
);
            fullQueue.setTitle("QUEUE | DEPARTURE");
            anchor.getChildren().addAll(logo, journeyLabel, dateLabel, hbox,
okBtn);
            Scene scene = guiElements.scene(anchor, 1000, 300, "style2.css");
            fullQueue.setScene(scene);
            fullQueue.showAndWait();
        }

        trainQueue.display();                                   //method that
displays a queue
        Thread.sleep(1000);
        viewPassengers(journey, localDate, window);             //view method
is called as the gui used is same
    }
```

```
        ////////////////////////////////////////////////////////////*****DELETE
METHOD*****////////////////////////////////////////////////////////////
    private void deletePassengers(String journey, LocalDate localDate, Stage
window){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Passenger's Name: ");
        String userInputName = sc.nextLine().toLowerCase().trim();
        System.out.println("Which Seat No: ");
//get user input
        String userSeat = sc.nextLine().toLowerCase().trim();
        System.out.println("Enter your NIC: ");
        String nic = sc.nextLine().trim();
        boolean flag = false;

        //for loop that loops through the entire queueArray checking for the
presence of the input name, nic and input seat; and if they are pointing at
        //the same passenger. If this condition satisfies, we loop again till
the last element (EXCLUDING) from the index (INCLUDING) at where the
        // condition was satisfied. All the values in between are assigned the
value which is at index + 1 from their actual index; signifying
        //forward movement - the index where the condition was satisfied is
overwritten, signifying deletion
        for (int i=0;i<trainQueue.getQueueArray().length;i++){
            if (userInputName.equals(trainQueue.getQueueArray()[i].getName())
&& nic.equals(trainQueue.getQueueArray()[i].getNIC())){
                if
(userSeat.equals(String.valueOf(trainQueue.getQueueArray()[i].getSeatsBooked()
))) {
                    flag = true;
//flag required for later
                    for (int j = i; j < trainQueue.getQueueArray().length;
j++) {
                        if (j != trainQueue.getQueueArray().length - 1) {
                            trainQueue.getQueueArray()[j] =
trainQueue.getQueueArray()[j + 1];
                        }
                    }
                }

            //this section is for: if NIC and seat numbers were correct but
not the name, the record is still deleted, as the unique identifier is NIC
            }else if (nic.equals(trainQueue.getQueueArray()[i].getNIC())){
                userInputName = trainQueue.getQueueArray()[i].getName();
                if
(userSeat.equals(String.valueOf(trainQueue.getQueueArray()[i].getSeatsBooked()
))){
                    flag = true;
                    for (int j=i;j<trainQueue.getQueueArray().length;j++) {
                        if (j != trainQueue.getQueueArray().length-1) {
                            trainQueue.getQueueArray()[j] =
trainQueue.getQueueArray()[j + 1];
                        }
                    }
                }
            }
        }
```

```
        //this section is responsible for updating the queue and it's values
        //once the swapping process is done, the length decreases by one, as a
result the index of the last element decreases by 1 too
        if (flag){
            queueLength--;
            int length = trainQueue.getLength() - 1;
            int last = trainQueue.getLast() - 1;
            trainQueue.setLength(length);
            trainQueue.setLast(last);
            System.out.println("Name : " + userInputName + " | Journey : " +
journey + " | Date : " + localDate  + " | Seat Number : " +
                    userSeat + " has been successfully deleted");
        }else{
            System.out.println("Name: " + userInputName + " hasn't booked seat
number: " + userSeat);
        }                                       //details of deleted passenger
outputted, if not found, appropriate message printed

        trainQueue.setUpdatedQueue();           //queue is updated after the
changes have been done

        try {
            displayMenu(journey, localDate, window);
        } catch (InterruptedException e) {
            System.out.println("something went WRONG");
        }
    }


    ////////////////////////////////////////////////////////////*****SAVE
METHOD*****////////////////////////////////////////////////////////////////
    private void savePassengers(String journey, LocalDate localDate, Stage
window){
        ////////////////////////////////////////////////////GETTING CLIENT
AND DATABASES//////////////////////////////////////////////////////////
        CodecRegistry pojoCodecRegistry =
fromRegistries(MongoClientSettings.getDefaultCodecRegistry(),

fromProviders(PojoCodecProvider.builder().automatic(true).build()));
        MongoClientSettings settings =
MongoClientSettings.builder().codecRegistry(pojoCodecRegistry).build();
        MongoClient mongoClient = MongoClients.create(settings);
//create mongoClient and get database
        MongoDatabase database = mongoClient.getDatabase("TrainQueue");
//directly saving POJO's weren't possible so settings had to change
        database.getCollection(localDate + " " + journey + " " + "Queue-
WaitingRoom").drop();
        database.getCollection(localDate + " " + journey + " " + "Queue-
OnTrain").drop();
        database.getCollection(localDate + " " + journey + " " + "Queue-
QueueArray").drop();    //the databases are dropped before saving as
        database.getCollection(localDate + " " + journey + " " + "Queue-
Variables").drop();     //it is required to overwrite instead of append

        MongoCollection<Passenger> collection =
database.getCollection(localDate + " " + journey + " " + "Queue-WaitingRoom",
Passenger.class);
        MongoCollection<Passenger> collection2 =
database.getCollection(localDate + " " + journey + " " + "Queue-OnTrain",
Passenger.class);
```

```
        MongoCollection<Passenger> collection3 =
database.getCollection(localDate + " " + journey + " " + "Queue-QueueArray",
Passenger.class);
        MongoCollection<Document> collection4 =
database.getCollection(localDate + " " + journey + " " + "Queue-Variables");
        //the collections are then obtained, this creates the collection to if
the collection isn't there

        ////////////////////////////////////////////////////////SAVING THE
VARIABLES////////////////////////////////////////////////////////
        //a document is created for the queue variables, for queue and report,
which are put into it with a key and the variable as a value
        Document firstPIndex = new Document();
        firstPIndex.put("First Passenger", firstPersonIndex);
        firstPIndex.put("First Q Variable", trainQueue.getFirst());
        firstPIndex.put("Last Q Variable", trainQueue.getLast());
        firstPIndex.put("Actual Length Q Variable", trainQueue.getLength());
        firstPIndex.put("Report First Current", newPassengerIndexForReport);
        firstPIndex.put(" Queue Length", queueLength);
        firstPIndex.put("Queue Max Length Attained",
trainQueue.getMaxLength());
        firstPIndex.put("Least time", trainQueue.getLeastTime());
        firstPIndex.put("Most time", trainQueue.getMaxTimeInQueue());
        firstPIndex.put("AVG time", trainQueue.getAvgTime());
        firstPIndex.put("Passenger count for AVG time",
trainQueue.getPassengerCount());
        firstPIndex.put("Total time for AVG time", trainQueue.getTotalTime());

        ////////////////////////////////////////////////////SAVING INTO THE
COLLECTIONS////////////////////////////////////////////////////////
        //a dummy passenger object as it isn't possible to save empty lists,
this is required if for instance the waiting room was empty when we save.
        //The list onTrain and array queueArray don't become empty so it isn't
required
        if (waitingRoom.isEmpty()) {
            waitingRoom.add(new Passenger());
        }
        collection.insertMany(waitingRoom);

        collection2.insertMany(onTrain);            //the arraylists are
directly added into each collection, while the queue array is added, after
        collection3.insertMany(Arrays.asList(trainQueue.getQueueArray()));
//being converted into an arraylist
        collection4.insertOne(firstPIndex);         //as for the queue
variables the document they've been appended to is added into it's collection

        try {
            displayMenu(journey, localDate, window);
        } catch (InterruptedException e) {
            System.out.println("something went WRONG");
        }
    }
```

```
        ////////////////////////////////////////////////////////////*****LOAD
METHOD*****////////////////////////////////////////////////////////////
    private void loadPassengers(String journey, LocalDate localDate, Stage
window) {
        ////////////////////////////////////////////////GETTING CLIENT AND
DATABASES/////////////////////////////////////////////////////////
        CodecRegistry pojoCodecRegistry =
fromRegistries(MongoClientSettings.getDefaultCodecRegistry(),

fromProviders(PojoCodecProvider.builder().automatic(true).build()));
        MongoClientSettings settings =
MongoClientSettings.builder().codecRegistry(pojoCodecRegistry).build();
        MongoClient mongoClient = MongoClients.create(settings);
//create mongoClient and get database
        MongoDatabase database = mongoClient.getDatabase("TrainQueue");
//directly saving POJO's weren't possible so settings had to change

        //getting all the data from the databases; getting their corresponding
iterables, and removing off the "_id" fields
        MongoCollection<Passenger> collection =
database.getCollection(localDate + " " + journey + " " + "Queue-WaitingRoom",
Passenger.class);

        MongoCollection<Passenger> collection2 =
database.getCollection(localDate + " " + journey + " " + "Queue-OnTrain",
Passenger.class);
        MongoCollection<Passenger> collection3 =
database.getCollection(localDate + " " + journey + " " + "Queue-QueueArray",
Passenger.class);
        MongoCollection<Document> collection4 =
database.getCollection(localDate + " " + journey + " " + "Queue-Variables");
        FindIterable<Passenger> waiting = collection.find();
FindIterable<Passenger> train = collection2.find();
        FindIterable<Passenger> queue = collection3.find();
FindIterable<Document> variables = collection4.find();
        waiting.projection(exclude("_id")); train.projection(exclude("_id"));
        queue.projection(exclude("_id"));
variables.projection(exclude("_id"));

        ////////////////////////////////////////////////////////LOADING OF THE
VARIABLES/////////////////////////////////////////////////////////
        //adding all the variable values into a document
        try {
            Document documentVariables = new Document();
            for (Document docVar : variables) {
                documentVariables = docVar;
            }

            //looping through the keyset and adding the values into a
temporary list
            List<Integer> requiredVariables = new ArrayList<>();
            for (String id : documentVariables.keySet()) {
                requiredVariables.add((Integer) documentVariables.get(id));
            }
```

```
        //the variables are then set their respective values from the
temporary list
            trainQueue.setTotalTimeForLoad(requiredVariables.get(11));
            trainQueue.setPassengerCountForLoad(requiredVariables.get(10));
            trainQueue.setAvgTimeForLoad(requiredVariables.get(9));
            trainQueue.setMaxTimeInQueueForLoad(requiredVariables.get(8));
            trainQueue.setLeastTimeForLoad(requiredVariables.get(7));
            trainQueue.setMaxLengthForLoad(requiredVariables.get(6));
            queueLength = requiredVariables.get(5);
            newPassengerIndexForReport = requiredVariables.get(4);
            trainQueue.setLength(requiredVariables.get(3));
            trainQueue.setLast(requiredVariables.get(2));
            trainQueue.setFirst(requiredVariables.get(1));
            firstPersonIndex = requiredVariables.get(0);

            ////////////////////////////////////////////LOADING OF THE
PASSENGER ARRAY LISTS//////////////////////////////////////////////////////
            //each corresponding iterable, pointing to each collection, is
iterated through added into a temporary list and is set to the corresponding
            //arrayLists used of the queue class

        //loading content of the waiting room list
            List<Passenger> tempWaiting = new ArrayList<>();
            boolean hasContent = false;
            for (Passenger passenger : waiting) {
                if (passenger.getSeatsBooked() != 0) {
                    tempWaiting.add(passenger);
                    hasContent = true;
                }
            }
            //if only there's actual content do we assign it to the waiting
room, otherwise we clear it
            if (hasContent) {
                waitingRoom = tempWaiting;
            } else {
                waitingRoom.clear();
            }

            //loading content of the onTrain list
            List<Passenger> tempOnTrain = new ArrayList<>();
            for (Passenger passenger : train) {
                tempOnTrain.add(passenger);
            }
            onTrain = tempOnTrain;

            //loading content of the actual passenger queue
            List<Passenger> tempQueue = new ArrayList<>();
            for (Passenger passenger : queue) {
                tempQueue.add(passenger);
            }
            trainQueue.setQueueArray(tempQueue.toArray(new Passenger[0]));
            trainQueue.setUpdatedQueue();
        }catch (Exception E){
            System.out.println("There is nothing to LOAD");
        }
```

```
        try {
            displayMenu(journey, localDate, window);
        } catch (InterruptedException e) {
            System.out.println("something went WRONG");
        }
    }

    ////////////////////////////////////////////////////////*****RUN
SIMULATION****////////////////////////////////////////////////////////
    private void runSimulation(String journey, LocalDate localDate, Stage
window) throws InterruptedException{
        ////////////////////////////////////////////////**SIMULATION
FUNCTIONALITY**////////////////////////////////////////////////////////
        //variables pointing to the process delay; most, least and avg times
spent in the queue
        int processDelay, maxLengthAttained = 0, mostTimeInQueue = 0,
leastTimeInQueue = 0, avgTimeInQueue = 0;

        //looping till all passengers have been added
        while (!trainQueue.getUpdatedQueue().isEmpty()) {
            processDelay = 0;                                   //process delay for
each passenger initializing

            trainQueue.setMaxLength();
            maxLengthAttained = trainQueue.getMaxLength(); //assigning the max
length of queue value for the report

            //for loop that loops three times, produces 3 random values from
6-sided die and adds them into the total process delay, per passenger
            for (int i = 0; i < 3; i++) {
                processDelay += new Random().nextInt(6) + 1;
            }
            trainQueue.setPassengerCountForLoad(trainQueue.getPassengerCount()
+ 1);
            //increment the counter variable, for average time, of the queue
class by one for every loop

            //we assign on train index at 0 to the passenger we removed by the
method remove(), the index is then incremented so we can add the next
            //passenger and so on
            onTrain.set(firstPersonIndex, trainQueue.remove());
            firstPersonIndex++;
            //as seen in the view method we start from this index as well, so
whenever we increment this value the queue is updated with
            //the person up till that index not included, as such showing the
movement of the queue forward as the person at that index is added

            //for loop that sets the waiting time for all the passengers in
the queue including front
            for (Passenger passenger : trainQueue.getUpdatedQueue()) {
                passenger.setSecondsInQueue(processDelay);
            }

            //we add the time of the passenger who's in front to the total
time variable of the queue class, in order to obtain the average
            trainQueue.setTotalTimeForLoad(trainQueue.getTotalTime() +
trainQueue.getUpdatedQueue().get(0).getSecondsInQueue());
```

```
        //setting and getting the most, least and avg times spent in queue as
a whole
            trainQueue.setMaxTimeInQueue();
            mostTimeInQueue = trainQueue.getMaxTimeInQueue();
            trainQueue.setLeastTime();
            leastTimeInQueue = trainQueue.getLeastTime();
            trainQueue.setAvgTime();
            avgTimeInQueue = trainQueue.getAvgTime();
            Thread.sleep(1000);

         //this for loop is required to check the seconds of the passengers
who have boarded the train - whether is it less than/ more than the
            //least and most time variables, even though we check the current
queue and update, if there was for instance, a passenger with a more
            //suitable time for these variables who has already boarded the
train, then this loop will handle that
            for(Passenger passenger : onTrain){
                if (passenger.getName() != null) {
                    if (passenger.getSecondsInQueue() > mostTimeInQueue) {
                        mostTimeInQueue = passenger.getSecondsInQueue();
                    } else if (passenger.getSecondsInQueue() <
leastTimeInQueue) {
                        leastTimeInQueue = passenger.getSecondsInQueue();
                    }
                }
            }


////////////////////////////////////////////////////**INDIVIDUAL
REPORT**//////////////////////////////////////////////////////////
            Stage individualReportWindow = new Stage();

            if (onTrain.get(newPassengerIndexForReport).getName() != null) {
                HBox hbox = guiElements.hbox(10, 10, 150);
                ImageView logo =
guiElements.imageViewLay("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/
Pictures/LOGO1.png",
                        420, 0, 100, 200);
                Label journeyLabel = guiElements.labels(40, 20, "Journey: " +
journey, "jDLabels");
                Label dateLabel = guiElements.labels(800, 20, "Date:  " +
localDate, "jDLabels");
                Label details = guiElements.labels(60, 370, "⚠\tMax Length
of Queue : \t" + maxLengthAttained +
                        "\t|\tMost Time Spent : \t" + mostTimeInQueue +
"\t|\tLeast Time Spent : \t" + leastTimeInQueue +
                        "\t|\tAverage Time Spent : \t" + avgTimeInQueue +
"\t⚠", "lengthLbl");              //passenger info, and the queue details
                Label passengerInfo = guiElements.labels(10, 150, "Passenger
Name : " +
                        onTrain.get(newPassengerIndexForReport).getName() +
"\tSeat Booked : " + onTrain.get(newPassengerIndexForReport).getSeatsBooked()
                        + "\t\tSeconds in Queue : " +
onTrain.get(newPassengerIndexForReport).getSecondsInQueue(), "individualRep");
                passengerInfo.setMinWidth(980);
                Button okBtn = guiElements.buttons("CONTINUE", 450, 250,
"trainViewBtn");
                AnchorPane anchor = guiElements.anchor();
                hbox.getChildren().add(passengerInfo);
```

```
//////////////////////////////////////////////////////////////GUI
ELEMENTS//////////////////////////////////////////////////////////

                ////////////////////////////////////////////////////WINDOW
ON CLOSE REQUEST//////////////////////////////////////////////////////////
                okBtn.setOnAction(event -> individualReportWindow.close());
                individualReportWindow.setOnCloseRequest(event -> {
                    event.consume();
                    individualReportWindow.close();
                });

                ////////////////////////////////////////////////////////SCENE
CREATION//////////////////////////////////////////////////////////
                individualReportWindow.getIcons().add(new
Image("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/Pictures/icon.png")
);
                individualReportWindow.setTitle("QUEUE | DEPARTURE");
                anchor.getChildren().addAll(logo, journeyLabel, dateLabel,
details, hbox, okBtn);
                Scene scene = guiElements.scene(anchor, 1000, 400,
"style2.css");
                individualReportWindow.setScene(scene);
                individualReportWindow.showAndWait();

                //info about passenger added
                System.out.println("Passenger Name: " +
trainQueue.getQueueArray()[newPassengerIndexForReport].getName() +
                        " | Seat Number: " +
trainQueue.getQueueArray()[newPassengerIndexForReport].getSeatsBooked() +
                        " has been added");
                newPassengerIndexForReport++;  //a different index value, for
the currently to be added person, (we cannot use the first person index
            }                                   //here since it's incremented
before we show the alert, and would show the person after the actual one)
            Thread.sleep(1000);

            //the queue is updated at the end, cuz the first passenger
wouldn't be considered otherwise
            trainQueue.setUpdatedQueue();
        }

        ////////////////////////////////////////////////////////**COMPLETE
REPORT GUI**//////////////////////////////////////////////////////////
        ImageView logo =
guiElements.imageViewLay("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/
Pictures/LOGO1.png",
                525, 0, 100, 300);
        Label journeyLabel = guiElements.labels(20, 20, "Journey: " + journey,
"jDLabels");
        Label dateLabel = guiElements.labels(1190, 20, "Date:  " + localDate,
"jDLabels");
        Label headers = guiElements.labels(140, 70, "Name \t \t \t \t \t Seats
Booked \t \t \t \t \t Time in Queue/s", "headerLbl");
        Label details = guiElements.labels(103, 610, "⚠\tMax Length of Queue
: \t" + maxLengthAttained +
                "\t \t \t|\t \tMost Time Spent : \t" + mostTimeInQueue + "\t
\t  |\t \t \tLeast Time Spent : \t" + leastTimeInQueue +
                "\t \t \t|\t \tAverage Time Spent : \t" + avgTimeInQueue +
"\t⚠", "lengthLbl");
```

```
        Button okBtn = guiElements.buttons("CONTINUE", 620, 640,
"trainViewBtn");
        VBox trainLabels = guiElements.vbox(10, 200, 130);
        ScrollPane scroll = guiElements.reportScroll(1080, 480, 140, 120, "-
fx-background-color: #222; " +
                "-fx-background:#222; -fx-border-color: #fff;");
        AnchorPane anchor = guiElements.reportAnchor();
        Label emptyLabel = new Label();
        boolean flag = false;
        ////////////////////////////////////////////////////////////////GUI
ELEMENTS////////////////////////////////////////////////////////////////

        ////////////////////////////////////////LABELS OF EACH PASSENGERS T0
BE ADDED INTO THE SCROLL PANE//////////////////////////////////////////
        //this for loop, loops through the passengers currently on the train
and adds them into the trainLabels vBox
        for (Passenger passenger : onTrain) {
            if (passenger.getName() != null) {
                Label label;
                if (passenger.getName().length()<9) {
                    label = new Label("\t" + passenger.getName() + "\t \t \t
\t \t \t \t \t \t \t \t \t" + passenger.getSeatsBooked() +
                            "\t \t \t \t \t \t \t \t \t \t \t \t \t" +
passenger.getSecondsInQueue());
                }else{
                    label = new Label("\t" + passenger.getName() + "\t \t \t
\t \t \t \t \t \t \t \t \t" + passenger.getSeatsBooked() +
                            "\t \t \t \t \t \t \t \t \t \t \t \t \t" +
passenger.getSecondsInQueue());
                }
                label.setId("runPassLabel");
                trainLabels.getChildren().add(label);
                flag = true;
            }
        }

        //an appropriate label, if there aren't any passenger currently on the
train
        if (!flag) {
            emptyLabel = guiElements.labels(240, 300, "⚠NO ONE HAS
BOARDED⚠", "emptyReport");
        }
        scroll.setContent(trainLabels);     //adding the labels into the
scrollPane

        ////////////////////////////////////////////////////////FILE DETAILS
HANDLER////////////////////////////////////////////////////////////////
        //wrapped in a try-catch block to catch IOExceptions
        try {
            FileWriter reportFile = new FileWriter("Report Details " + journey
+ " " + localDate);
            reportFile.write("*****Currently in Train*****\n");
```

```
        //loop through passengers currently on train and add their details
onto the txt file at the end after simulation
            for (Passenger passenger : onTrain) {
                if (passenger.getName() != null) {
                    reportFile.write("*Passenger Name: " + passenger.getName()
+ "\n Seat Booked: " + passenger.getSeatsBooked()
                            + "\n Seconds in Queue: " +
passenger.getSecondsInQueue() + "\n Most time Spent in Queue of a Passenger: "
+
                            mostTimeInQueue + "\n Least time Spent in Queue of
a Passenger: " + leastTimeInQueue +
                            "\n AVG time Spent in Queue: " + avgTimeInQueue +
"\n\n");
                }
            }
            reportFile.close();


        } catch (IOException e) {
            System.out.println("An error occurred in creating the file");
        }

        ////////////////////////////////////////////////////////WINDOW CLOSE
REQUEST//////////////////////////////////////////////////////////////
        okBtn.setOnAction(event -> {
            event.consume();
            window.close();
            try {
                displayMenu(journey, localDate, window);
            } catch (InterruptedException e) {
                System.out.println("something went WRONG");
            }
        });

        window.setOnCloseRequest(event -> {
            event.consume();
            closeScenes(journey, localDate, window);
        });

        ////////////////////////////////////////////////////////////SCENE
CREATION//////////////////////////////////////////////////////////////
        anchor.getChildren().addAll(logo, journeyLabel, dateLabel, scroll,
headers, details, emptyLabel, okBtn);
        Scene scene = guiElements.scene(anchor, 1366, 705, "style2.css");
//adding all elements to the container
        window.setScene(scene);
//connecting css files, setting initial scene to display
        window.show();
//currently boarded window, and showing it
    }
```

```
        ////////////////////////////////////////////////////////////*****WINDOW
ALERT HANDLER*****////////////////////////////////////////////////////////////
    private void closeScenes(String journey, LocalDate localDate, Stage
window) {
        Alert closeAlert = guiElements.closeWindowCommon();      //alert pop up
upon closing of the gui window of the add and view methods
        closeAlert.initOwner(window);                            //this is only
to get the icon of the window to show up on the alerts
        closeAlert.showAndWait();
        if (closeAlert.getResult() == ButtonType.YES) {
            window.close();                                      //if clicked
on yes the window is closed, and the menu is displayed again
            try {
                displayMenu(journey, localDate, window);
            } catch (InterruptedException e) {
                System.out.println("something went WRONG");      //in a try-
catch block due to the timing functions running
            }
        } else {
            closeAlert.close();
        }
    }

    public static void main(String[] args) {launch(args);}
}
```

PassengerQueue class

```
package sample;

import java.util.ArrayList;
import java.util.List;

public class PassengerQueue {
    private Passenger[] queueArray = new Passenger[42];          //array and
arrayList for entire queue array; queue elements ONLY that updates
    private List<Passenger> updatedQueue = new ArrayList<>();
    private int first, last, length;                             //first, last
positions of queue, length is actual length of queue with passengers

    private int maxLength, leastTime, maxTimeInQueue, avgTime;  //for the
report
    public int passengerCount, totalTime; //for AVG time calculation
    //max length is the max length recorded in the queue, most, least, avg
times are for all queue passengers, total time is total spent by everyone
    //in queue, passenger count is total number of passengers who have joined
queue - to get avg time

    //add method, to add people from waiting room into queue
    public void add(Passenger next){
        if (!isFull()){                                 //only do the process if the
queue isn't currently full
            queueArray[last] = next;
            length++;                                   //here we are basically
adding passengers to the end
            last++;
        }
    }

    //remove method to remove front person from queue
    public Passenger remove(){
        Passenger frontPassenger = queueArray[first];
        if(!isEmpty()){                                 //return removed passenger
to be used in Station class - representing the person added onto train
            first++;
            length--;                                   //we move queue forward by
incrementing "first" instance variable
        }else{
            System.out.println("Queue is currently empty");
        }
        return frontPassenger;
    }

    //setters and getters for all the instance variables of position at first,
last, length and max length, least, most and avg times
    public void setFirst(int first) { this.first = first; }
    public int getFirst(){
        return first;
    }

    public void setLast(int last){ this.last = last; }
    public int getLast(){
        return last;
    }
```

```
    public void setLength(int length){
        this.length = length;
    }
    public int getLength() { return length; }


    //this section is for the variables to be used for the report, setters and
getters for those. In addition a secondary setter is created as for the
    //loading process
    public void setMaxLength(){
        if (this.getLength()>this.getMaxLength()){
            this.maxLength = this.getLength();
        }
    }
    public int getMaxLength(){
        return this.maxLength;
    }
    public void setMaxLengthForLoad(int maxLength){
        this.maxLength = maxLength;
    }

    public void setLeastTime(){
        if (!getUpdatedQueue().isEmpty()) {
            this.leastTime = queueArray[0].getSecondsInQueue();     //setting
an initial value
            for (Passenger passenger : getUpdatedQueue()) {
                if (this.leastTime > passenger.getSecondsInQueue()) {
                    this.leastTime = passenger.getSecondsInQueue();
                }
            }
        }
    }
    public int getLeastTime(){
        return this.leastTime;
    }
    public void setLeastTimeForLoad(int leastTime){
        this.leastTime = leastTime;
    }

    public void setMaxTimeInQueue(){
        for (Passenger passenger : getUpdatedQueue()){
            if (passenger.getSecondsInQueue()>this.maxTimeInQueue){
                this.maxTimeInQueue = passenger.getSecondsInQueue();
            }
        }
    }
    public int getMaxTimeInQueue(){ return this.maxTimeInQueue; }
    public void setMaxTimeInQueueForLoad(int maxTimeInQueue){
        this.maxTimeInQueue = maxTimeInQueue;
    }

    public void setAvgTime(){
        if (totalTime != 0) {
            this.avgTime = totalTime / passengerCount;
        }
    }
    public int getAvgTime(){
        return this.avgTime;
    }
```

```
    public void setAvgTimeForLoad(int avgTime){
        this.avgTime = avgTime;
    }

    public void setPassengerCountForLoad(int passengerCount) {
        this.passengerCount = passengerCount;
    }
    public int getPassengerCount() { return passengerCount; }

    public int getTotalTime() { return totalTime; }
    public void setTotalTimeForLoad(int totalTime) { this.totalTime =
totalTime; }

    //setting and returning the updated queue as an arrayList - only the
values
    public void setUpdatedQueue(){
        updatedQueue.clear();
        for (int i=0;i<length;i++){
            updatedQueue.add(queueArray[(first+i)%42]);
        }
    }
    public List<Passenger> getUpdatedQueue(){return updatedQueue;}

    //setting the queue array, this is needed for the load method, to
initialize the queue array
    public void setQueueArray(Passenger[] newQueue){
        queueArray = newQueue;
    }
    public Passenger[] getQueueArray(){return queueArray;}


    //booleans returners, if queue is empty/ full
    public boolean isEmpty(){
        return getLength()==0;
    }

    //returning whether queue is full
    public boolean isFull(){
        return getLength()==42;
    }

    //simply printing out the current queue
    public void display(){
        System.out.print("Seats: ");
        for (int i=0;i<length;i++){
            System.out.print(queueArray[(first+i)].getSeatsBooked() + " - " +
queueArray[(first+i)].getName() + " ");
        }
        System.out.println();
    }
}
```

Passenger Class

```
package sample;

public class Passenger {
    private String name, NIC;                //instance variables of names,
seconds, and seats booked of a passenger
    private int secondsInQueue, seatsBooked;

    //sets and returns names for all the passengers
    public void setName(String name){
        if (name.trim().equals("")){
            System.out.println("Invalid Name");     //if there's an invalid
name print message, otherwise set instance
        }else{                                      //variable to that
variable
            this.name = name;
        }
    }
    public String getName(){
        return name;
    }

    public void setNIC(String NIC){ this.NIC = NIC; }
    public String getNIC(){ return NIC; }

    //sets and returns seats booked for a passenger
    public void setSeatsBooked(int seatsBooked){
        this.seatsBooked = seatsBooked;
    }
    public int getSeatsBooked(){
        return seatsBooked;
    }       //"this" refers to the current instance variable of the object
created; necessary only when the name of parameter and itself are the same

    //sets and returns seconds in queue for a passenger
    public void setSecondsInQueue(int secondsInQueue){
        this.secondsInQueue += secondsInQueue;
    }
    public int getSecondsInQueue(){
        return secondsInQueue;
    }
}
```

GuiElements Class

```
////////////////////////////////////////////////////////////GUI ELEMENTS
CLASS/////////////////////////////////////////////////////////////////////

package sample;

import javafx.geometry.Pos;
import javafx.scene.Cursor;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;                    //importing necessary fx nodes
import javafx.stage.Stage;

import java.time.LocalDate;

public class GuiElements {
    DropShadow shadow = new DropShadow();       //object of DropShadow class
for button effects

    //anchorPane node, for the containers
    public AnchorPane anchor(){
        AnchorPane anchor = new AnchorPane();
        anchor.setStyle("-fx-background-color: #222; -fx-border-color:
#f00;");
        return anchor;
    }

    //an alternate anchor pane used for the report, takes id, layouts and min
sizes.
    public AnchorPane reportAnchor(){
        AnchorPane anchorPane = new AnchorPane();
        anchorPane.setId("reportPane");
        return anchorPane;
    }

    //the scroll pane, layout for the final report gui, which displays the
details of people in the queue
    public ScrollPane reportScroll(int width, int height, int layX, int layY,
String style){
        ScrollPane scrollPane = new ScrollPane();
        scrollPane.setVbarPolicy(ScrollPane.ScrollBarPolicy.AS_NEEDED);
        scrollPane.setPrefSize(width, height);
        scrollPane.setStyle(style);
        scrollPane.setLayoutX(layX);
        scrollPane.setLayoutY(layY);
        return scrollPane;
    }
```

```
        //label node, holds parameters for layout x, y text content and its id
values
    public Label labels(int layX, int layY, String text, String id){
        Label labels = new Label();
        labels.setLayoutX(layX);
        labels.setLayoutY(layY);
        labels.setText(text);
        labels.setId(id);
        labels.setAlignment(Pos.CENTER);
        return labels;
    }

    //textField node, holds parameters for layout x, y text content, height
and width values
    public TextField textFields(int layX, int layY, int height, int width,
String prompt){
        TextField textField = new TextField();
        textField.setLayoutX(layX);
        textField.setLayoutY(layY);
        textField.setMinHeight(height);
        textField.setMinWidth(width);
        textField.setPromptText(prompt);
        return textField;
    }

    //ImageView node, holds parameters for layout x, y, height, width and the
image file location values
    public ImageView imageViewLay(String imageFile, int layX, int layY, int
height, int width){
        Image imageLay = new Image(imageFile);
        ImageView imageViewLay = new ImageView(imageLay);
        imageViewLay.setFitHeight(height);
        imageViewLay.setFitWidth(width);
        imageViewLay.setX(layX);
        imageViewLay.setY(layY);
        return imageViewLay;
    }

    //button node, holds parameters for layout x, y text content and its id
values, further holds the shadow as an effect and cursor hover effect
    public Button buttons(String btnText, int layX, int layY, String id){
        Button button = new Button();
        button.setText(btnText);
        button.setLayoutX(layX);
        button.setLayoutY(layY);
        button.setId(id);
        button.setCursor(Cursor.HAND);
        button.setEffect(shadow);
        return button;
    }
```

```
        //ComboBox node, for the datePicker and journey gui, holds the list of
journeys x and y layouts, default value and min width
    public ComboBox<String> allJourneys(){
        ComboBox<String> allJourneys =new ComboBox<>();
        allJourneys.setOnMouseEntered(event ->
allJourneys.setCursor(Cursor.HAND));
        allJourneys.getItems().addAll("Colombo - Badulla", "Colombo -
Polgahawela", "Colombo - Peradeniya Junction",
                "Peradeniya Junction - Nanuoya", "Peradeniya Junction -
Badulla", "Colombo - Gampola", "Colombo - Nawalapitiya",
                "Colombo - Hatton", "Colombo - Thalawakele", "Colombo -
Nanuoya", "Nanuoya - Badulla", "Colombo - Haputale", "Colombo - Diyatalawa",
                "Colombo - Bandarawela", "Colombo - Ella", "Badulla - Ella",
"Badulla - Bandarawela", "Badulla - Diyatalawa",
                "Badulla - Haputale", "Badulla - Nanuoya", "Nanuoya -
Colombo", "Nanuoya - Peradeniya Junction", "Peradeniya Junction - Colombo",
                "Badulla - Thalawakele", "Badulla - Hatton", "Badulla -
Nawalapitiya", "Badulla - Gampola", "Badulla - Peradeniya Junction",
                "Peradeniya Junction - Colombo", "Badulla - Polgahawela",
"Badulla - Maradana", "Badulla - Colombo");
        allJourneys.setLayoutX(220);
        allJourneys.setLayoutY(250);
        allJourneys.setMinWidth(400);
        allJourneys.setValue("Colombo - Badulla");
        return allJourneys;
    }

    //datePicker node, with disabled editing ability to prevent entering of
fake text, and disabling of past dates, setting default date, x y layouts,
    //min widths
    public DatePicker datePicker(){
        DatePicker datePicker = new DatePicker();
        datePicker.setOnMouseEntered(event ->
datePicker.setCursor(Cursor.HAND));
        datePicker.getEditor().setDisable(true);
        datePicker.setDayCellFactory(picker -> new DateCell(){
            public void updateItem(LocalDate date, boolean empty){
                super.updateItem(date, empty);
                LocalDate today = LocalDate.now();
                setDisable(empty || date.compareTo(today)<0);
            }
        });
        datePicker.setLayoutX(220);
        datePicker.setLayoutY(170);
        datePicker.setMinWidth(400);
        datePicker.setValue(LocalDate.now());
        return datePicker;
    }

    //hBox node, holds parameters for layout x and spacing between elements in
the container
    public HBox hbox(int spacing, int layX, int layY){
        HBox layoutSeats = new HBox(spacing);
        layoutSeats.setLayoutX(layX);
        layoutSeats.setLayoutY(layY);
        return layoutSeats;
    }
```

```
        //vBox node, holds parameters for layout x, y and spacing between
elements in the container
    public VBox vbox(int spacing, int layX, int layY){
        VBox vbox = new VBox(spacing);
        vbox.setLayoutX(layX);
        vbox.setLayoutY(layY);
        vbox.setAlignment(Pos.CENTER);
        return vbox;
    }

    //gridPane node, holds parameters for layout x, further has default vGap,
hGap and layout y values for elements in the container
    public GridPane gridPane(int layX){
        GridPane gridPane = new GridPane();
        gridPane.setLayoutX(layX);
        gridPane.setLayoutY(160);
        gridPane.setHgap(20);
        gridPane.setVgap(5);
        return gridPane;
    }

    //tilePane node, holds default vGap, hGap and layout y and x values for
elements in the container
    public TilePane tilePane(int layX, int layY){
        TilePane layoutTrain = new TilePane();
        layoutTrain.setVgap(25);
        layoutTrain.setHgap(50);
        layoutTrain.setLayoutY(layY);
        layoutTrain.setLayoutX(layX);
        return layoutTrain;
    }

    //scene node, which will take the anchor pane, a width height for the
window, and the file for the css
    public Scene scene(AnchorPane anchor, int width, int height, String file){
        Scene scene = new Scene(anchor, width, height);
        String css = this.getClass().getResource(file).toExternalForm();
        scene.getStylesheets().add(css);
        return scene;
    }

    //Alert node, which is common for the confirmation alerts to be used, has
a custom graphic, button types and a header text and title
    public Alert closeWindowCommon(){
        ImageView imageConfirm =
imageViewLay("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/Pictures/per
son.png",
                0, 0, 100, 100);
        Alert closeAlert = new Alert(Alert.AlertType.CONFIRMATION, null,
ButtonType.YES, ButtonType.NO);
        closeAlert.setHeaderText("Do you REALLY want to exit?");
        closeAlert.setGraphic(imageConfirm);
        closeAlert.setTitle("Denuwara Menike Intercity");
        return closeAlert;
    }
```

```
        //method that handles the action done onto the alert from the above
method, for the date gui
    public void closeDateGui(Stage window){
        Alert closeAlert = closeWindowCommon();
        closeAlert.showAndWait();
        if (closeAlert.getResult()==ButtonType.YES) {
            window.close();
        }else{
            closeAlert.close();
        }
    }

    //error alert for the train booking add method
    public void errorAlert(String text){
        ImageView imageConfirm =
imageViewLay("file:/C:/Users/Ammuuu/Downloads/W1761196/coursework/Pictures/err
or.png",
                0, 0, 100, 100);
        Alert closeAlert = new Alert(Alert.AlertType.ERROR, null,
ButtonType.OK);
        closeAlert.setHeaderText(text);
        closeAlert.setGraphic(imageConfirm);
        closeAlert.setTitle("Denuwara Menike Intercity");
        closeAlert.showAndWait();
        closeAlert.close();
    }
}
```

ADD SEATS

TEST CASES	OUTPUT
Enter A, passengers are sent from waiting room into the train queue	Entering A each time sends the next set of passengers into the train queue
Random amount generated through a 6-sided dice	Random amounts of passengers are sent each time each is added depending on a 6-sided dice
Once loaded waiting room passengers are displayed	Once loaded, waiting room passengers are displayed along with their name and corresponding seat number
Passengers are added without entering passenger data	Entering of A adds the passengers, no need of entering any kind of details
Train queue displayed with proper arrangements	Upon entering A each time, GUI opens up and the train queue is displayed, with a difference in color, and name and seat number against slot
Display error when queue is full	This particular test case doesn't get to be tested as the queue is assumed to be of size 42. However, once all the passengers have joined the queue entering of A will display the message.
GUI must always be updated	the next time add is called, the train queue is updated

VIEW TRAIN QUEUE

TEST CASES	OUTPUT
Entering of V displays train queue	Enter V on GUI, displays train queue GUI, with 42 slots
Train queue GUI accurate	Each passengers name beside their seat numbers, if they are currently in the queue, else if not, a label saying "empty" is displayed alongside the slot
Train seating arrangement GUI	Shifting scenes to the train seating GUI displays the train seating
Train seating GUI accurate	If passenger has arrived, before joining the queue, his name is labelled beside seat number, and once they enter the train the border color of the button changes signifying on board
Waiting room arrangement GUI	Shifting scenes to the waiting room GUI displays passengers in waiting room
Waiting room GUI accurate	If passenger arrived, they are sent to waiting room, here you can visualize passengers currently in waiting room along with their seat numbers (signified further with red bordering)
GUI must always be updated	Whenever the queue or train has been updated the GUI is updated as well

DELETE PASSENGER FROM QUEUE

TEST CASES	OUTPUT
Entering of D requests user to enter passenger name	Passenger name, NIC and seat number are requested to be entered by user, since even-though, passenger can book multiple seats, still there'll be a different passenger for each seat
Entering of correct details deletes passenger	Corresponding passenger is deleted and their details are shown
Train queue is ordered properly afterwards	Upon deleting of a customer train queue is re-ordered properly, with each passenger, from the deleted passenger's position, brought front
Entering of incorrect name but correct seat number and NIC would still delete	If for instance there is a mix up with same names, in real life, entering of a dummy name along with correct NIC would still delete the corresponding passenger

SAVE DATA

TEST CASES	OUTPUT
Data saved into MongoDB upon entering S	Entering S on the console saves data of the current queue, waiting room, on train, onto the database, along with the instance variables required for the queue class
Upon exiting of program, data must be retained	Data is retained even after closing of the program

LOAD DATA

TEST CASES	OUTPUT
Data is retrieved upon loading	Data saved is brought back into the corresponding data structures and instance variables
GUI is therefore to be updated	GUI will therefore, revert to how it was before closing of program
Any changes done upon loading, saving and retrieval of that data must update the GUI over and over again	The GUI is updated accordingly to the data saved. As the file saved is overwritten each time any changes that have done

RUN SIMULATION

TEST CASES	OUTPUT
Entering R on console runs the simulation	The simulation runs
Passenger currently at the start of the queue added into the train	The passenger who is at the beginning index is added into the train from the queue
A random process delay is generated for each passenger	A random process delay is generated by a total of 3 6-sided die
This process delay is added to all the passengers in the queue	The process delay is added to each passenger's secondsInQueue instance variable
Report of the passenger is generated	A pop-up GUI comes up containing their details, max, least and avg times along with the max length of the queue
The time values and max length is always updated	The values are updated accordingly to the queue passengers
The details of the passengers are to be saved in txt file	The passenger's details, who joined the train, are written into a file writer text file
GUI must always be updated	Upon entering R the queue GUI and on train GUI's of add and view methods are updated
Runs until all passengers in queue have been added into train	Pop-up GUI of each passenger is shown until all passengers have joined the train (the train queue is empty)
Details in the report are accurate	The queue length, max and least times are accurate, as for the average time, it is accurate but is a value rounded down to that of the actual value

VALIDATIONS

TEST CASES	OUTPUT
Prompt display option until Q entered	Menu is continuously prompted until Q is entered
Each option calls a different method	Each option calls a different method. A V calling the add and view methods, S L D R the save, load, delete and run methods respectively
Invalid data type entry/ a wrong option is caught without displaying an error	Any other input other than A, V, S, L, D, R is caught throwing an error message
Closing of GUI during A, V or E doesn't quit program	Clicking on 'X' on the window prompts a confirmation alert to the user, if it was a mistake user can return back into the GUI, if It wasn't the GUI is closed and the menu is called again
Validation for lowercase option entry	The respective methods are called regardless of case-sensitivity.
In delete entering of incorrect details not in the data structure mustn't throw an error.	An appropriate message is outputted if for incorrect details which isn't in the data structure has been referred to.
Whitespace entry mustn't be taken into consideration	Once the customer name input is taken in any of these methods the trailing whitespaces are removed
Exceptions mustn't throw errors and stop execution	Try-catch blocks are there in each place that might throw errors (InterruptedException for example). As for null pointers, there are empty passenger objects in areas where there aren't actual passengers.
Don't allow continuation of initial GUI without a journey and Date chosen	Editing of the Date field is disabled, and a default Date and journey have already been set to prevent this.
If a passenger has reserved many seats deleting of him should only delete a specified seat not all of his seats	Even though a passenger can book many, there are still that many passengers, so the name along with seat number is requested to delete only a passenger's specific seat
A null passenger object mustn't be considered during the process	There are checks that whether the name if each passenger is not null, if only will the execution occur (waiting room has 42 passenger objects, but might have fewer actual passengers; doing this prevents null pointers)
Upon starting the program, the correct data must be loaded into passengers	Upon starting, the details in CW-01 are loaded, accurately
Waiting room must have data accordingly to whether passenger has arrived or not	After the initial check, on whether each seat number has arrived, the waiting room is updated.
Shouldn't throw an error if R is hit before any passenger has joined the queue	If R is entered before any passenger has joined, an empty label is displayed on the GUI

EXTRAS

TEST CASES	OUTPUT
NoSQL used	NoSQL has been used as the data storage and retrieval method using MongoDB as the database
Naming conventions followed	Naming conventions have been followed. With variables having meaning, methods named as verbs and the class as a noun.
Alert boxes are validated	The Alert boxes are validated for each choice respectively. For instance, if the window closing confirmation alert returns NO the control is returned back into the GUI.
The report GUI generated must have accurate values	The values and details shown in the final report are accurate, taking into consideration everyone who was ever in the queue.