

Informatics Institute of Technology

In collaboration with

University Of Westminster

Programming Principles II

4COSC003W

**Course Work - Two**

**Assingment Documentation-Report**

Module Leader’s Name – Mr. Guhanathan Poravi

Student Name: Nuwin (Hansitha) Godakanda Arachchi

UOW Number: W1761350

IIT Student ID: 2019443

**Contents**

[Introduction 3](#_Toc38744834)

[Implementation and Output 4](#_Toc38744835)

[Test Plan 98](#_Toc38744837)

[Self-Assessment 101](#_Toc38744838)

[Technical Discussion 103](#_Toc38744839)

[Conclusion 103](#_Toc38744840)

# **Introduction**

This assignment documentation paper refers to the program file available on the corresponding zip-file. The coed is a fully functional program code, developed using Java SDK 8.2, while using its Design Property, JavaFX for its designing portions.

The complete performance of this code has been systematically gone through, throughout this paper including descriptive information. Further in, test plans collected and found results of using black-box and white-box testing have also been attached in this paper. This program can be identified as a extenstion to the coursework-one paper refered by the same author. Thus, similarities may exist throughout the paper.

Owned by the Sri Lanka Railways Department, Denuwara Menike is a train that travels through miraculous sceneries and fantastic cityscapes that you’ve got to see to believe. With its special design to walk through the heavy hills of the hill county in Sri Lanka, Denuwara Menike starts its voyage from Colombo-Fort, the commercial capital of Sri Lanka, and goes along until Badulla, close to the cental hills of Sri Lanka, Kandy. We software engineering students of Univeristy of Westminster were challanged to build a code to run a simulate of what will really happen at a train station.

The program this paper refers is a fully functional program written for the afore mentioned specification. The program allows the following functionality:

* Allows to load data directly from previous coursework, notified above;
* Add and remove data from the train queuel;
* Delete a passenger from the queue at any given moment;
* Gather time and queue length statistics for both queues.

All the functions are fully described throughout this paper.

Enjoy.

# **Implementation and Output**

This section of the paper mainly focuses on the functionality of the code. All the methods, alongside its outputs.

Check In method – checkIn()

Function check in is where the user can confirm his/her presence at the train station to board to the train. First things first, this function allows the station master or an office from the railways department to select the direction of the train and the current station they currently are in since my code for coursework one allows to account for multiple stops.

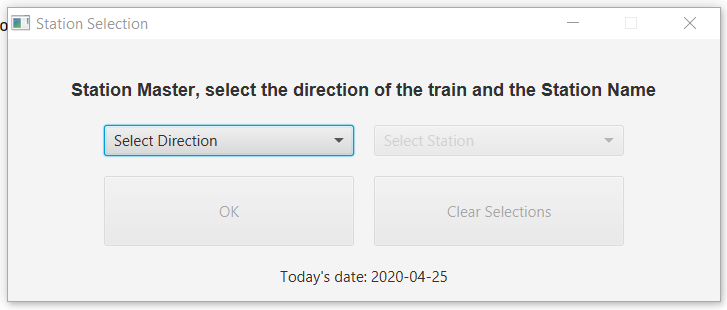
Once the station and the direction is selected, the officer can open up a window with shows all the passengers who start their journeys at the selected station. These names will be followed up by a set o buttons allowing the user to click on them to check in. Once clicked, the program will get a confirmation from the user to confirm he/she is selecting the right check-in field.

After a specific time, the officer can close the check-in portal. This would restrict passengers from joining later however. But considering an airport being the same, I’ve implemented the close check-in button.

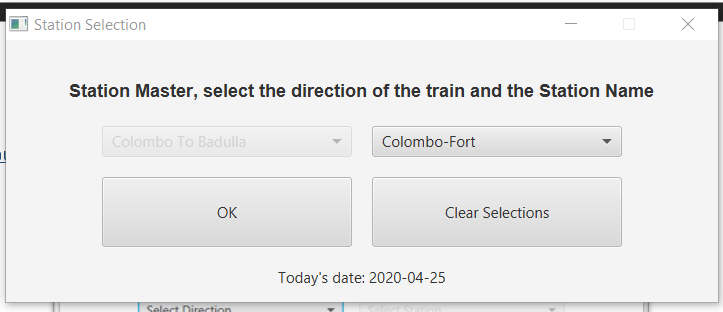
Source Code- Function Check In

public static void checkIn() {  
  
 //============================================================================================================//  
 // INITIALIZING ELEMENTS  
  
 //------------------------------------------------------------------------------------------------------- Stages  
  
 ArrayList<String> newList = new ArrayList<>();  
  
 Stage stageOne = new Stage();  
 BorderPane rootOne = new BorderPane();  
 Scene sceneOne = new Scene(rootOne,700,250);  
 stageOne.setScene(sceneOne);  
 rootOne.getStylesheets().add("/style.css");  
 stageOne.setResizable(false);  
 stageOne.setTitle("Station Selection");  
 stageOne.show();  
  
 Stage stageTwo = new Stage();  
 BorderPane rootTwo = new BorderPane();  
 Scene sceneTwo = new Scene(rootTwo,1000,800);  
 stageTwo.setScene(sceneTwo);  
 rootTwo.getStylesheets().add("/style.css");  
 stageTwo.setResizable(false);  
 stageTwo.setTitle("Denuwara Menike Terminal- Self Check In");  
 stageTwo.initStyle(StageStyle.UNDECORATED); // Making the window undecorated so that the check in box cannot be  
 // cancelled unwillingly  
  
 LocalDate date = LocalDate.now(); //getting current date  
  
 //------------------------------------------------------------------------------------------------------- Labels  
 Label mainLabel = new Label("Denuwara Menike Terminal- Check-In Window");  
 mainLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 30));  
 mainLabel.setPadding(new Insets(40,20,20,20));  
  
 Label subLabel = new Label("Please select the 'Check In' Button co-responding with your seat");  
 subLabel.setFont(Font.font("sans-serif", FontPosture.REGULAR, 18));  
 subLabel.setPadding(new Insets(0,20,20,20));  
  
 Label stationMaster = new Label("Station Master, select the direction of the train and the Station Name");  
 stationMaster.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 18));  
 stationMaster.setPadding(new Insets(40,20,5,20));  
  
  
 Label stationMasterFooter = new Label("Today's date: "+date);  
  
 Label deets = new Label("Denuwara Menike Train Terminal; Station Name: "+stops[station]+  
 "; Date: "+date);  
  
  
 //------------------------------------------------------------------------------------------------------ Buttons  
  
 Button stationSubmit = new Button("OK");  
 stationSubmit.setId("stationButtons");  
  
 Button stationClear = new Button("Clear Selections");  
 stationClear.setId("stationButtons");  
  
 Button closeCheckIn = new Button("Close Check In");  
 closeCheckIn.setId("closeCheckIn");  
  
 //-------------------------------------------------------------------------------------------------- Combo Boxes  
  
 ComboBox selectDirection = new ComboBox();  
 selectDirection.setId("combo");  
 selectDirection.getItems().addAll("Colombo To Badulla", "Badulla to Colombo");  
 selectDirection.setPromptText("Select Direction");  
  
 ComboBox selectStation = new ComboBox();  
 selectStation.setId("combo");  
 selectStation.setPromptText("Select Station");  
  
  
  
 //-------------------------------------------------------------------------------------------------- Alert Boxes  
  
 Alert confirmClose = new Alert(Alert.AlertType.CONFIRMATION);  
 confirmClose.setHeaderText("Confirm Close");  
 confirmClose.setTitle("Are you sure to close the Check In counter?");  
 confirmClose.setContentText("Closing this will not allow anymore people to check in");  
  
 Alert confirmCheckIn = new Alert(Alert.AlertType.CONFIRMATION);  
 confirmCheckIn.setTitle("Confirm Check In");  
 confirmCheckIn.setContentText("Make sure you've selected the correct Check-In box");  
  
 //============================================================================================================//  
 // GUI INITIALIZATION  
  
 //--------------------------------------------------------------------------------------Stage One GUI Components  
  
 //-----Main boxes used  
 VBox headerOne = new VBox(20);  
 HBox headerLineOne = new HBox(20);  
 HBox footerLineOne = new HBox(20);  
  
 //-----Alignment of the main boxes  
 headerOne.setAlignment(Pos.CENTER);  
 headerLineOne.setAlignment(Pos.CENTER);  
 footerLineOne.setAlignment(Pos.CENTER);  
  
 rootOne.setTop(headerOne);  
  
 //-----putting in labels into the header  
 headerOne.getChildren().add(stationMaster);  
 headerOne.getChildren().add(headerLineOne);  
 headerOne.getChildren().add(footerLineOne);  
 headerOne.getChildren().add(stationMasterFooter);  
  
 headerLineOne.getChildren().add(selectDirection);  
 headerLineOne.getChildren().add(selectStation);  
 footerLineOne.getChildren().add(stationSubmit);  
 footerLineOne.getChildren().add(stationClear);  
  
 //-----default disabling elements  
 selectStation.setDisable(true);  
 stationSubmit.setDisable(true);  
 stationClear.setDisable(true);  
  
  
 //--------------------------------------------------------------------------------------Stage Two GUI Components  
  
 //-----Main boxes used  
 VBox headerTwo = new VBox(10);  
 HBox centerTwo = new HBox(75);  
 VBox footerTwo = new VBox(20);  
  
 // creating a scroll pane to display the names  
 ScrollPane centerScroll = new ScrollPane(centerTwo);  
  
 VBox names = new VBox(30);  
 VBox ids = new VBox(30);  
 VBox buttons = new VBox(20);  
  
 //-----Initializing elements  
 names.setPadding(new Insets(14,0,0,0));  
 ids.setPadding(new Insets(18,0,0,0));  
 buttons.setPadding(new Insets(14,0,0,0));  
 centerScroll.setPadding(new Insets(12,0,12,0));  
  
 rootTwo.setTop(headerTwo);  
 rootTwo.setCenter(centerScroll);  
 rootTwo.setBottom(footerTwo);  
  
 headerTwo.setAlignment(Pos.CENTER);  
 centerTwo.setAlignment(Pos.CENTER);  
 footerTwo.setAlignment(Pos.CENTER);  
  
 names.setAlignment(Pos.BASELINE\_RIGHT);  
 ids.setAlignment(Pos.CENTER);  
  
 centerScroll.setContent(centerTwo);  
 centerScroll.setFitToWidth(true);  
  
 headerTwo.getChildren().add(mainLabel);  
 headerTwo.getChildren().add(subLabel);  
  
 centerTwo.getChildren().add(names);  
 centerTwo.getChildren().add(ids);  
 centerTwo.getChildren().add(buttons);  
  
 for (int i=0;i<=12;i++){  
 selectStation.getItems().add(stops[i]); // populating comboBox using array- stops  
 }  
  
 selectDirection.valueProperty().addListener((observable, oldValue, newValue) -> {  
  
 selectStation.setDisable(false); //disabling already selected buttons  
 selectDirection.setDisable(true);  
 stationClear.setDisable(false);  
  
 selectStation.valueProperty().addListener(((observable1, oldValue1, newValue1) -> {  
 for (int i=0;i<=12;i++){  
 if (stops[i].equals(newValue1.toString())){ //traversing array to get selected value index  
 station=i; //storing index of value selected  
 }  
  
 if (newValue.toString().contains("Colombo To Badulla")){  
 direction="ctb"; //initializing global variables according to inputs  
 }else{  
 direction="btc";  
 }  
 }  
  
 stationSubmit.setDisable(false);  
  
 }));  
 });  
  
 stationSubmit.setOnAction(event -> {  
 stageOne.close();  
 stageTwo.show();  
  
 System.out.println("Getting your data from the System. Hold Tight!");  
 MongoClient client = MongoClients.create();  
 MongoDatabase dataBase = client.getDatabase("BookingDB");  
 MongoCollection<Document> baseCollection = dataBase.getCollection("bookCollection");  
 Document tempHold = baseCollection.find().first();  
 Object seats = tempHold.get("name");  
 seatList = (ArrayList<String>) seats;  
 System.out.println("Data retrieval Successful!");  
  
 for (int i = seatList.size() - 1; i >= 0; i--) { //checking for old data and deleting them  
 String current = seatList.get(i).substring(13, 23);  
 String curDirection = seatList.get(i).substring(3, 6);  
 int curStation = Integer.parseInt(seatList.get(i).substring(7,9));  
 String strDate = date.toString();  
 if (current.compareTo(strDate) != 0 || (!(curDirection.equals(direction))) || (curStation!=station)) {  
 seatList.remove(i);  
 }  
 }  
  
 Button[] checkIn = new Button[seatList.size()];  
  
 for (int i=0;i<seatList.size();i++){  
 // creating labels of data in order or visual ease  
 //retrieving data from concatted strings received from database  
 String fullName= seatList.get(i).substring(24,seatList.get(i).length()-5);  
 String id= seatList.get(i).substring(seatList.get(i).length()-4);  
 String[] name=fullName.split(" ", 2);  
 String firstName= name[0].substring(0,1).toUpperCase()+name[0].substring(1);  
 String secondName= name[1].substring(0,1).toUpperCase()+name[1].substring(1);  
  
 //creating buttons and labels, and putting values into them  
 names.getChildren().add(new Label(firstName+" "+secondName));  
 ids.getChildren().add(new Label(id));  
 checkIn[i]=new Button(" Click to Check In");  
 checkIn[i].setId("checkInButtons");  
 buttons.getChildren().add(checkIn[i]);  
 }  
  
 for (int i=0;i<seatList.size();i++){  
 int finalI = i;  
  
 checkIn[i].setOnAction(event1 -> { // set on action for each check in button  
  
 String wholeName = seatList.get(finalI).substring(24, seatList.get(finalI).length() - 5);  
 String uniqueID = seatList.get(finalI).substring(seatList.get(finalI).length() - 4);  
 String seatNumber = seatList.get(finalI).substring(0,2);  
 String[] splitNames = wholeName.split(" ", 2);  
  
 // changing the header text area to name of passenger for easy verification  
 confirmCheckIn.setHeaderText(wholeName);  
  
 Optional<ButtonType> result = confirmCheckIn.showAndWait(); // confirm check in request  
 if (result.get() == ButtonType.OK) {  
 checkIn[finalI].setDisable(true);  
 checkIn[finalI].setText("Checked In- Successful");  
  
 newList.add(seatList.get(finalI));  
 seatList.set(finalI, "null");  
 waitingRoom[Integer.parseInt(seatNumber)] = new Passenger(splitNames[0], splitNames[1], uniqueID, seatNumber);  
 //upon confirming, will be added to waiting list  
  
 }  
  
 });  
 }  
 });  
  
 //getting exit buttons and footer information to footer  
 footerTwo.getChildren().add(closeCheckIn);  
 footerTwo.getChildren().add(deets);  
  
 closeCheckIn.setOnAction(event -> {  
 Optional<ButtonType> result = confirmClose.showAndWait(); // confirming exit request  
 if (result.get() == ButtonType.OK) {  
 stageTwo.close(); //upon confirming, stage will be closed  
 try {  
 int count = 0;  
 for (int i=seatList.size()-1;i>=0;i--){  
 if (seatList.get(i).equals("null")){ //emptying seatlist  
 seatList.remove(i);  
 }else{  
 count++;  
 }  
 }  
 if (count>0) { // looking for number of passengers that didnt check in  
 System.out.println(count+" passenger(s) have not checked in.");  
 }else{  
 System.out.println("All passengers checked in!");  
 }  
 menu(); //calling back menu to sustain program- On press of exit button  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 stationClear.setOnAction(event -> {  
 stageOne.close(); //if clear selections button is pressed  
 try {  
 checkIn();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 });  
  
}

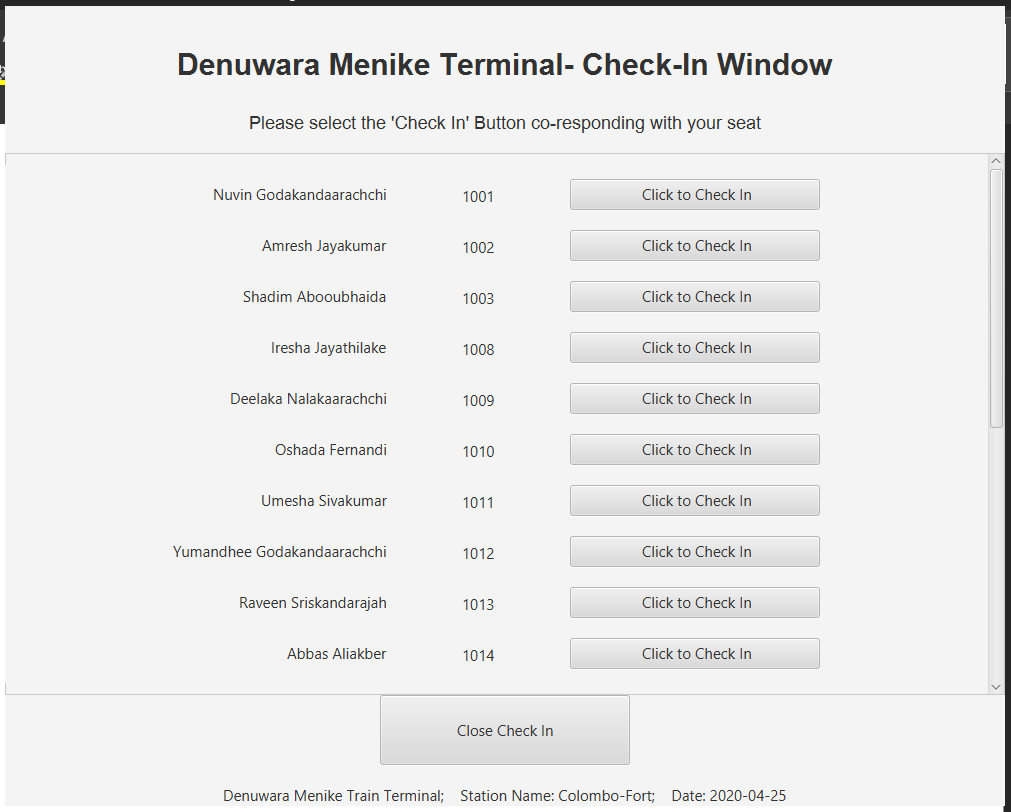
Output- Function Check In



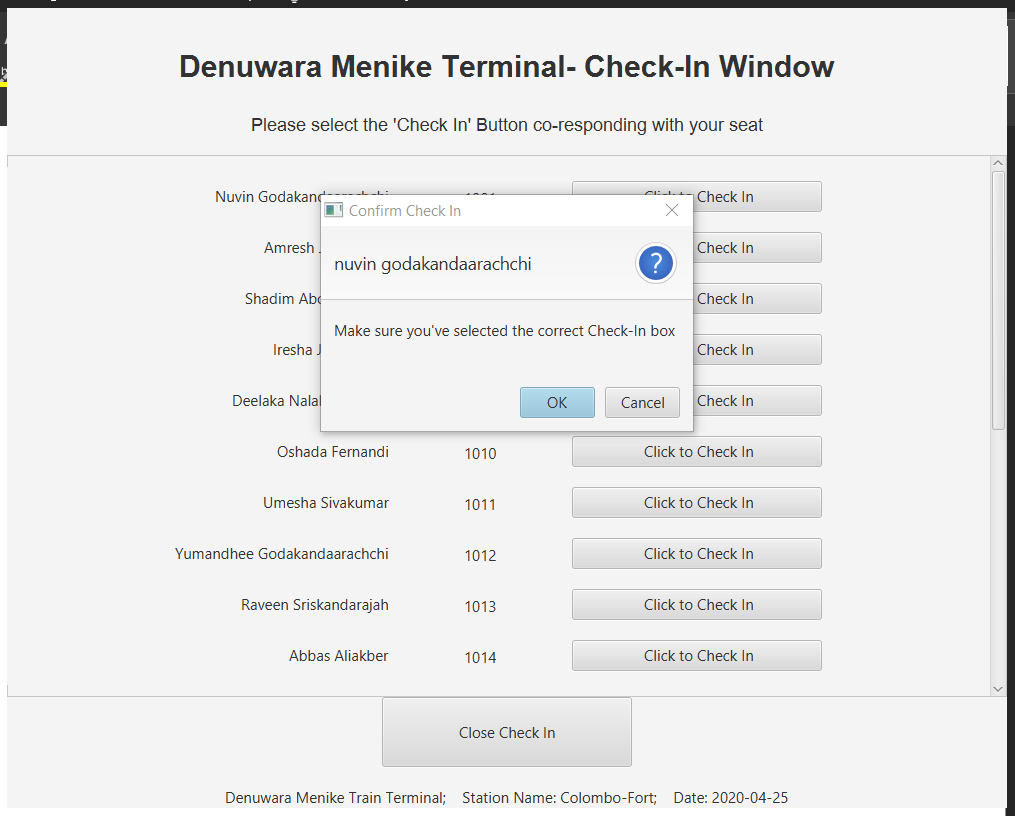
**Img 1.1-** The first stage, where the officer will be able to select the station and the direction



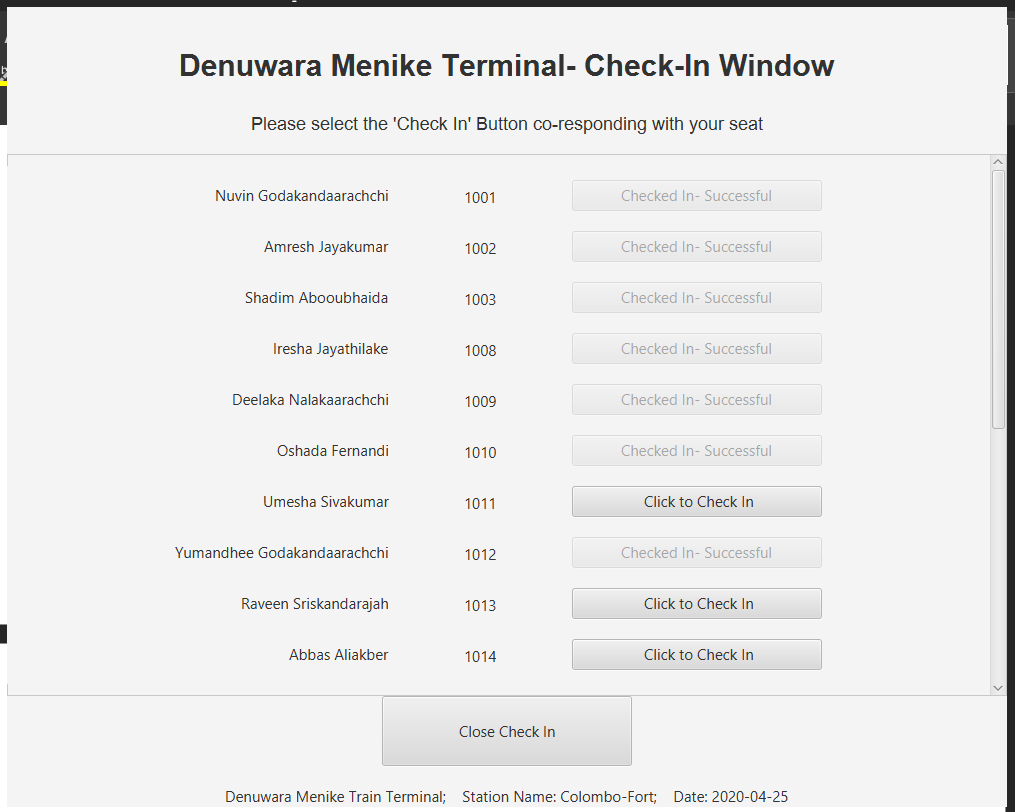
**Img 1.2-** After selection of the station in the giu, the ok button will be enabled



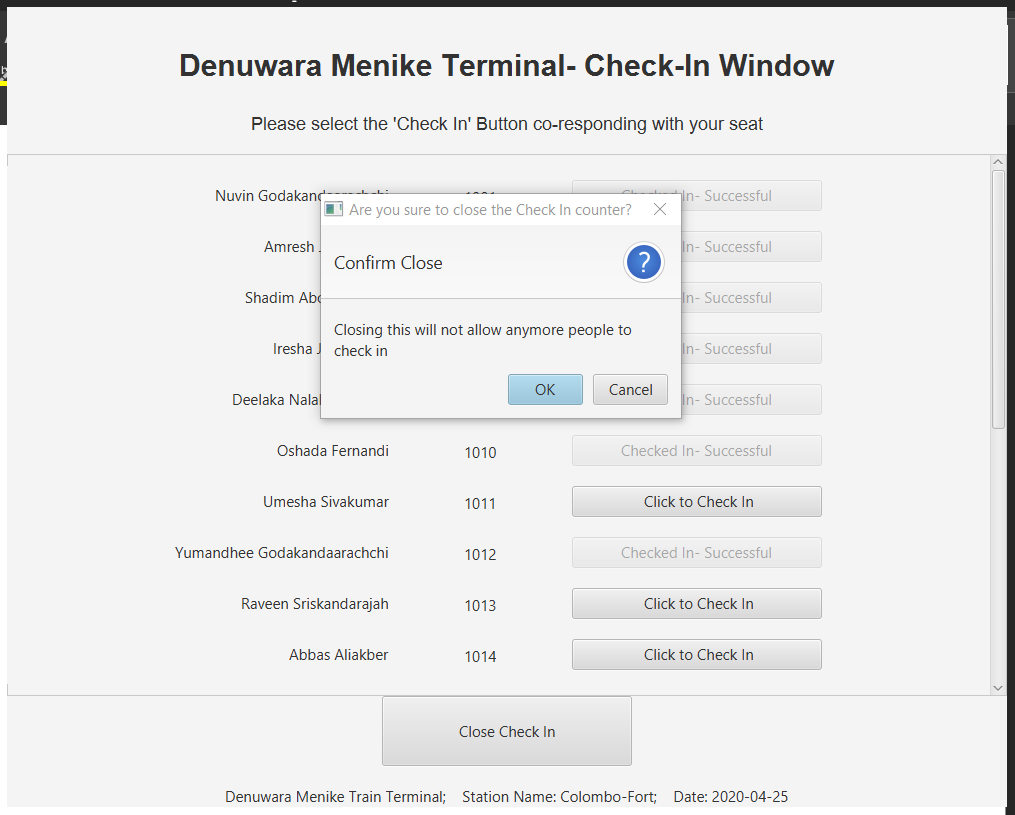
**Img 1.3-** Check in window that appears after selection of OK



**Img 1.4-** Once button “Check In” is clicked, confirmation window will pop



**Img 1.5-** Once several passengers have checked in



**Img 1.6-** Once close checkin button is selected

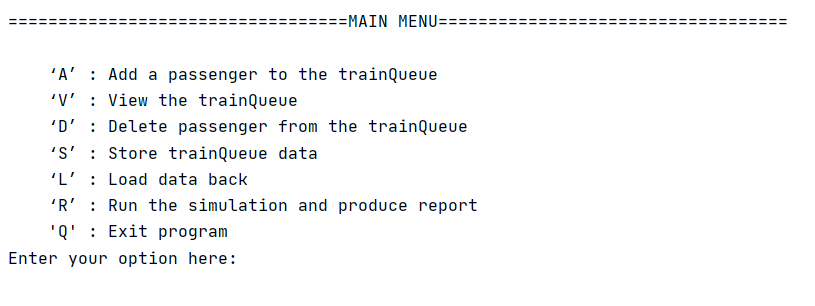
Menu method – menu()

Function Menu. This function acts as mother for all other methods. This is where, after the check-in counter, the program will direct the user to. Each method ends with a call to this method. Thus, proving its importance in the program.

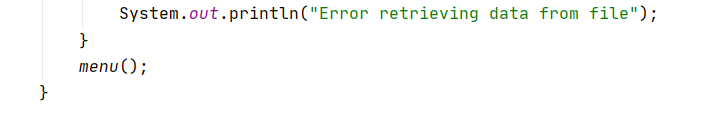
Source Code- Function Menu

public static void menu() {  
 Scanner input = new Scanner(System.in);  
 String option;  
 System.out.println(" ");  
 System.out.println("==================================MAIN MENU===================================");  
 System.out.println(" ");  
 System.out.println(" ‘A’ : Add a passenger to the trainQueue");  
 System.out.println(" ‘V’ : View the trainQueue");  
 System.out.println(" ‘D’ : Delete passenger from the trainQueue");  
 System.out.println(" ‘S’ : Store trainQueue data");  
 System.out.println(" ‘L’ : Load data back");  
 System.out.println(" ‘R’ : Run the simulation and produce report");  
 System.out.println(" 'Q' : Exit program");  
 System.out.print("Enter your option here: ");  
 option = input.next(); // receives input from user  
 option = option.toUpperCase();  
 switch (option) {  
 case "A":  
 addPassenger(); //call add customers and evokes GUI  
 break;  
 case "V":  
 view(); //call view all seats and evokes GUI  
 break;  
 case "D":  
 System.out.println("");  
 System.out.println("======================= Delete Passenger =======================");  
 System.out.println("");  
 delete(); //call delete customer data  
 break;  
 case "S":  
 System.out.println("");  
 System.out.println("======================= Store from File ========================");  
 System.out.println("");  
 store(); //call store data  
 break;  
 case "L":  
 System.out.println("");  
 System.out.println("======================== Load from Fle =========================");  
 System.out.println("");  
 load(); //call load data  
 break;  
 case "R":  
 System.out.println("");  
 System.out.println("======================== Run Simulation ========================");  
 System.out.println("");  
 run(); //call Simulaton  
 break;  
 case "Q":  
 System.out.println("Oh No! Are you sure to exit? Press <B> to cancel exit, or any other key to exit: ");  
 String choice= input.next().toLowerCase(); // getting users confirmation for exit  
 if (choice.equals("b")){ // if user chose to stay in program  
 menu();  
 }else { //if user chose to exit program  
 System.out.println("Thank you using Denuwara Manike Train Terminal System. " +  
 "Have a good day and a Safe Journey!");  
 System.exit(1);  
 }  
 default:  
 System.out.println("Oops! We couldn't read that. Please check the options again and re-enter.");  
 menu();  
 }  
}

Output- Function Menu



**Img 2.1-** How function menu will be output in the console



**Img 2.2-** Proof to that after each method, function menu will be called. Extracted from function load

# Add Passengers to the Train Queue (“A”) – addPassenger()

Function Add Passenger mainly focuses on adding passengers in the waiting room to the train queue. In this program, two queues have been used. Using one root hbox and two other vboxes, I have created this train queue.

The program allows the use of a random integer. The output from this randomizer will be the taken as the number of people who joined the train queue. In selecting a mechanism for the train queue selection, the program compares the queue with the least number of passengers in and put the passenger there. However, the program starts by adding the first value to the first stage. Thus, for the train queue one, the program allows to be checked if the values are equal.

If a train queue currently has no values, the label will be shown as a empty train queue in the queue diagram. However, as values get added, these default buttons disappear.

I have also created an additional waiting room display towards the right corner of the gui. This gui uses two flowpanes to position itself.

These seats in the waiting room will be originally of t a silver color, and later if the seat is occupied, turned to purple. When the user selects the button, a window will pop and show all the details of the passenger.

Source Code- Function Add Customers

public static void addPassenger(){  
  
 //============================================================================================================//  
 // INITIALIZING ELEMENTS  
  
 //------------------------------------------------------------------------------------------------------- Stages  
  
 Stage stage = new Stage();  
 BorderPane root = new BorderPane();  
 Scene scene = new Scene(root,1700,900);  
 stage.setScene(scene);  
 root.getStylesheets().add("/style.css");  
 stage.setResizable(false);  
 stage.setTitle("Denuwara Menike Terminal- ");  
  
 LocalDate date = LocalDate.now(); //local date  
  
 //---------------------------------------------------------------------------------------------------Alert Boxes  
  
 Alert noValues = new Alert(Alert.AlertType.WARNING);  
 noValues.setTitle("End of values");  
 noValues.setHeaderText("No more values to Add");  
 noValues.setContentText("No more values available to add to train queue");  
  
 Alert fullError = new Alert(Alert.AlertType.WARNING);  
 fullError.setTitle("Queue Full");  
 fullError.setHeaderText("Queue is filled up");  
 fullError.setContentText("The queue is already full");  
  
 //--------------------------------------------------------------------------------------------------------Labels  
  
 Label mainLabel = new Label("Denuwara Menike Terminal - Train Queue and Waiting Room");  
 mainLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 30));  
 mainLabel.setPadding(new Insets(60,20,5,20));  
  
 Label subLabel = new Label("Moving to function 'R' will move passengers to the train");  
 subLabel.setFont(Font.font("sans-serif", FontPosture.REGULAR, 18));  
 subLabel.setPadding(new Insets(5,20,5,20));  
  
 Label queueLabel = new Label("Train Queue Waiting to Board");  
 queueLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 22));  
 queueLabel.setPadding(new Insets(5,20,30,200));  
 queueLabel.setAlignment(Pos.CENTER);  
  
 Label waitingLabel = new Label("Waiting Room");  
 waitingLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 22));  
 waitingLabel.setPadding(new Insets(5,20,5,120));  
  
 Label queueOneLabel = new Label("Train Queue One");  
 queueOneLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,18));  
 queueOneLabel.setPadding(new Insets(0,0,5,0));  
  
 Label queueTwoLabel = new Label("Train Queue Two");  
 queueTwoLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,18));  
 queueTwoLabel.setPadding(new Insets(0,0,5,0));  
  
 Label deets = new Label("Denuwara Menike Train Terminal; Station Name: "+stops[station]+  
 "; Date: "+date);  
  
 //-------------------------------------------------------------------------------------------------------Buttons  
  
 Button exit = new Button(" Exit ");  
 exit.setId("closeExit");  
  
 //============================================================================================================//  
 // GUI INITIALIZATION  
  
 //------------------------------------------------------------------------------------------Stage GUI Components  
  
 //main boxes used  
 VBox header = new VBox();  
 VBox left = new VBox(5);  
 VBox center = new VBox();  
 VBox right = new VBox(5);  
 VBox footer = new VBox(10);  
  
 //-----------extra panes needed for both waiting room and queues  
  
 // used  
 VBox queueBoxOne = new VBox(5);  
 VBox queueBoxTwo = new VBox(5);  
  
 //used for displaying waiting room seats  
 FlowPane waitingSeatsOne = new FlowPane(15,15);  
 FlowPane waitingSeatsTwo = new FlowPane(15,15);  
  
 // setting boxes intitialized into box plot settings  
 root.setTop(header);  
 root.setLeft(left);  
 root.setCenter(center);  
 root.setRight(right);  
 root.setBottom(footer);  
  
 header.setAlignment(Pos.CENTER);  
 left.setAlignment(Pos.CENTER);  
 center.setAlignment(Pos.CENTER);  
 right.setAlignment(Pos.CENTER);  
 footer.setAlignment(Pos.CENTER);  
  
 FlowPane innerCenter = new FlowPane(); //used for waiting room gui  
 HBox innerLeft = new HBox();  
  
 innerLeft.setAlignment(Pos.CENTER);  
 innerCenter.setAlignment(Pos.CENTER);  
 queueBoxOne.setAlignment(Pos.CENTER);  
 queueBoxTwo.setAlignment(Pos.CENTER);  
  
 queueBoxOne.setPadding(new Insets(10, 0, 10, 200));  
 queueBoxTwo.setPadding(new Insets(10, 0, 10, 70));  
  
 //setting children  
 header.getChildren().add(mainLabel);  
 header.getChildren().add(subLabel);  
 left.getChildren().add(queueLabel);  
 left.getChildren().add(innerLeft);  
 center.getChildren().add(waitingLabel);  
 center.getChildren().add(innerCenter);  
 footer.getChildren().add(exit);  
  
 innerLeft.getChildren().add(queueBoxOne);  
 innerLeft.getChildren().add(queueBoxTwo);  
  
 innerCenter.getChildren().add(waitingSeatsOne);  
 innerCenter.getChildren().add(waitingSeatsTwo);  
  
 queueBoxOne.getChildren().add(queueOneLabel);  
 queueBoxTwo.getChildren().add(queueTwoLabel);  
  
 //--------------------------------------------------------------------------------- Program code controlling gui  
  
 Button[] queueButtons = new Button[20];  
  
 Random rand = new Random(); //importing random class  
 int num = rand.nextInt(6) + 1; // adding one to omit the occurance of o  
  
 for (int i=1;i<=num;i++){  
 boolean endLoop= false; //used to check when to exit the loop mentioned below  
  
 for (int count=0;count<42;count++){ //traverses whole array  
 int queueOne= trainQueueOne.getLength();  
 int queueTwo= trainQueueTwo.getLength();  
  
 //if condition checks if the specific value in the waiting room is not null  
 // also if it is not added9(according to the tag used to verify)  
 // train queue is not full  
 // also compares train queue sizes to enter values into the one with lesser passengers  
 if (waitingRoom[count]!=null && !waitingRoom[count].getAdded() && !trainQueueOne.isFull() &&  
 (queueTwo>queueOne || queueOne==queueTwo)){  
 trainQueueOne.add(waitingRoom[count]); //add the value from waiting room to train queue  
 waitingRoom[count].setAdded(true); //edits tag to true - ie. used  
 int diceOne= rand.nextInt(6)+1; // radomizing 3 more dice to get a random number to work as  
 int diceTwo= rand.nextInt(6)+1; // time taken  
 int diceThree= rand.nextInt(6)+1;  
 int time= diceOne+diceTwo+diceThree;  
 trainQueueOne.setTime(time); //using set time in the passenger queue class to set time  
 break;  
 }else if (waitingRoom[count]!=null && !waitingRoom[count].getAdded() && !trainQueueTwo.isFull() &&  
 queueOne>queueTwo){  
 trainQueueTwo.add(waitingRoom[count]); //add the value from waiting room to train queue  
 waitingRoom[count].setAdded(true); //edits tag to true - ie. used  
 int diceOne= rand.nextInt(6)+1; // radomizing 3 more dice to get a random number to work as  
 int diceTwo= rand.nextInt(6)+1; // time taken  
 int diceThree= rand.nextInt(6)+1;  
 int time= diceOne+diceTwo+diceThree;  
 trainQueueTwo.setTime(time); //using set time in the passenger queue class to set time  
 break;  
 }else if (trainQueueTwo.isFull() && trainQueueOne.isFull()){  
 System.out.println("All values added"); //if train queues are full , loop will be broken  
 break;  
 }else if(count==41){  
 endLoop=true;  
 noValues.showAndWait(); // if all values are added, messages will be prompted  
 break;  
 }  
 }  
 if (trainQueueTwo.isFull() && trainQueueOne.isFull()){  
 fullError.showAndWait(); // if both queues are full, again messages will be propmpted  
 break;  
 }  
 if (endLoop){  
 break; //if all values have been added loop will be broken again  
 }  
 }  
  
 // ---------------Train Queue gui program codes  
  
 if (trainQueueOne.isEmpty()){  
 Button emptyButton=new Button("Empty Train Queue"); // default buttons, sayaing empty  
 emptyButton.setId("queueButtons");  
 queueBoxOne.getChildren().add(emptyButton);  
 }else {  
 for (int i = 0; i < trainQueueOne.getLength(); i++) {  
 queueButtons[i] = new Button(trainQueueOne.accessName(i)); // names will be added to the button text  
 queueButtons[i].setId("queueButtons");  
 queueBoxOne.getChildren().add(queueButtons[i]);  
 }  
 }  
  
 if (trainQueueTwo.isEmpty()){  
 Button emptyButton=new Button("Empty Train Queue");// default buttons, sayaing empty  
 emptyButton.setId("queueButtons");  
 queueBoxTwo.getChildren().add(emptyButton);  
 }else {  
 for (int i = 0; i < trainQueueTwo.getLength(); i++) {  
 queueButtons[i] = new Button(trainQueueTwo.accessName(i));// names will be added to the button text  
 queueButtons[i].setId("queueButtons");  
 queueBoxTwo.getChildren().add(queueButtons[i]);  
 }  
 }  
  
 // --------------Waiting room giu program codes  
  
 Button[] waitingButtons = new Button[42];  
  
 //empty buttons firstly created  
 waitingSeatsOne.setPrefWrapLength(270);  
 waitingSeatsOne.setPadding(new Insets(70, 10, 10, 100));  
 for (int i=0;i<21;i++){  
 waitingButtons[i]=new Button("Empty"); // empty text  
 waitingButtons[i].setId("waitingButtons");  
 waitingSeatsOne.getChildren().add(waitingButtons[i]);  
 }  
  
 waitingSeatsTwo.setPrefWrapLength(270);  
 waitingSeatsTwo.setPadding(new Insets(70, 10, 10, 20));  
 for (int i=21;i<42;i++){  
 waitingButtons[i]=new Button("Empty"); //empty text  
 waitingButtons[i].setId("waitingButtons");  
 waitingSeatsTwo.getChildren().add(waitingButtons[i]);  
 }  
  
 // then will be colored  
 for (int i=0;i<42;i++){  
 if (waitingRoom[i]!=null && !waitingRoom[i].getAdded()){ //i+"\n Empty"  
 waitingButtons[Integer.parseInt(waitingRoom[i].getSeat())].setText(waitingRoom[i].getSeat()+"\n"+  
 waitingRoom[i].getName());  
 waitingButtons[Integer.parseInt(waitingRoom[i].getSeat())].setStyle("-fx-background-color: #c29ecd");  
 int finalI = i;  
 waitingButtons[i].setOnAction(event -> { // on click of these buttons will show a messsage box with all  
 // the passenger details  
 Alert info = new Alert(Alert.AlertType.INFORMATION);  
 info.setTitle("Passenger Details");  
 info.setHeaderText("Name: "+waitingRoom[finalI].getFullName());  
 info.setContentText("Seat Number: "+waitingRoom[finalI].getSeat()+"\n"+  
 "Unique ID: "+waitingRoom[finalI].getId());  
 info.showAndWait(); //showing the information box  
 });  
 }  
 }  
  
  
 stage.show();  
 footer.getChildren().add(deets);  
  
 exit.setOnAction(event -> { //closing the stage through exit button  
 stage.close();  
 menu();  
 });  
}

**Classes called in the PassengerQueue.java class:**

**Add: -** Used to add data to the queueArray inside the passengerQueue class. Described further below.

public void add (Passenger next){ //Setter  
 if (isFull()){  
 System.out.println("Train queue is full");  
 }else{  
 for (int numOne = 0; numOne < maxLength-1; numOne++) { // Using classical bubble sort  
 for (int numTwo = numOne + 1; numTwo < maxLength-1; numTwo++) {  
 // comparing adjacent strings  
 int one= Integer.parseInt(queueArray[numOne].getSeat());  
 int two= Integer.parseInt(queueArray[numTwo].getSeat());  
 if (two<one) {  
 Passenger cache = queueArray[numTwo];  
 queueArray[numTwo] = queueArray[numOne];  
 queueArray[numOne] = cache;  
 }  
 }  
 }  
 queueArray[last] = next;  
 last++;  
 maxLength++;  
 longestLength++;  
 }  
}

**SetTime:** Set the delay timer to each passenger

public void setTime(int time){  
 maxStayInQueue+=time;  
 queueArray[last-1].setSecondsInQueue(maxStayInQueue);  
  
 if (time>longestStay){  
 longestStay=time;  
 }  
 if (time<shortestStay){  
 shortestStay=time;  
 }  
}

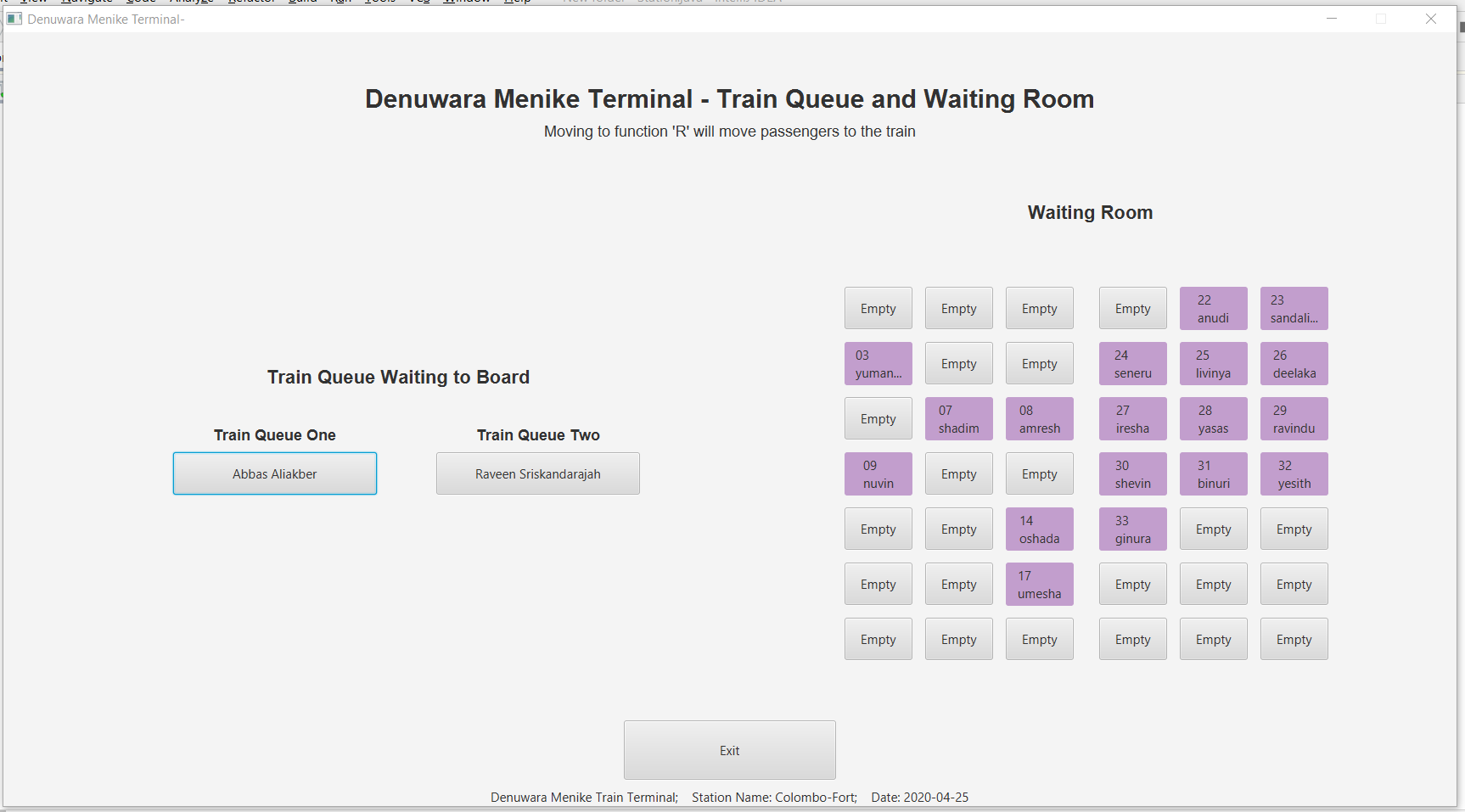
**getLength: -** Used to get length

public int getLength(){  
 return maxLength;  
}

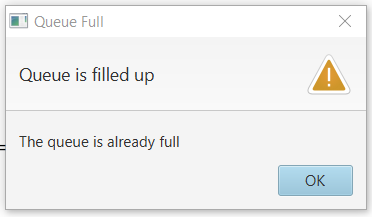
**AccessName:** Used to access the full name of a given index

public String accessName(int index){  
 return queueArray[index].getFullName();  
}

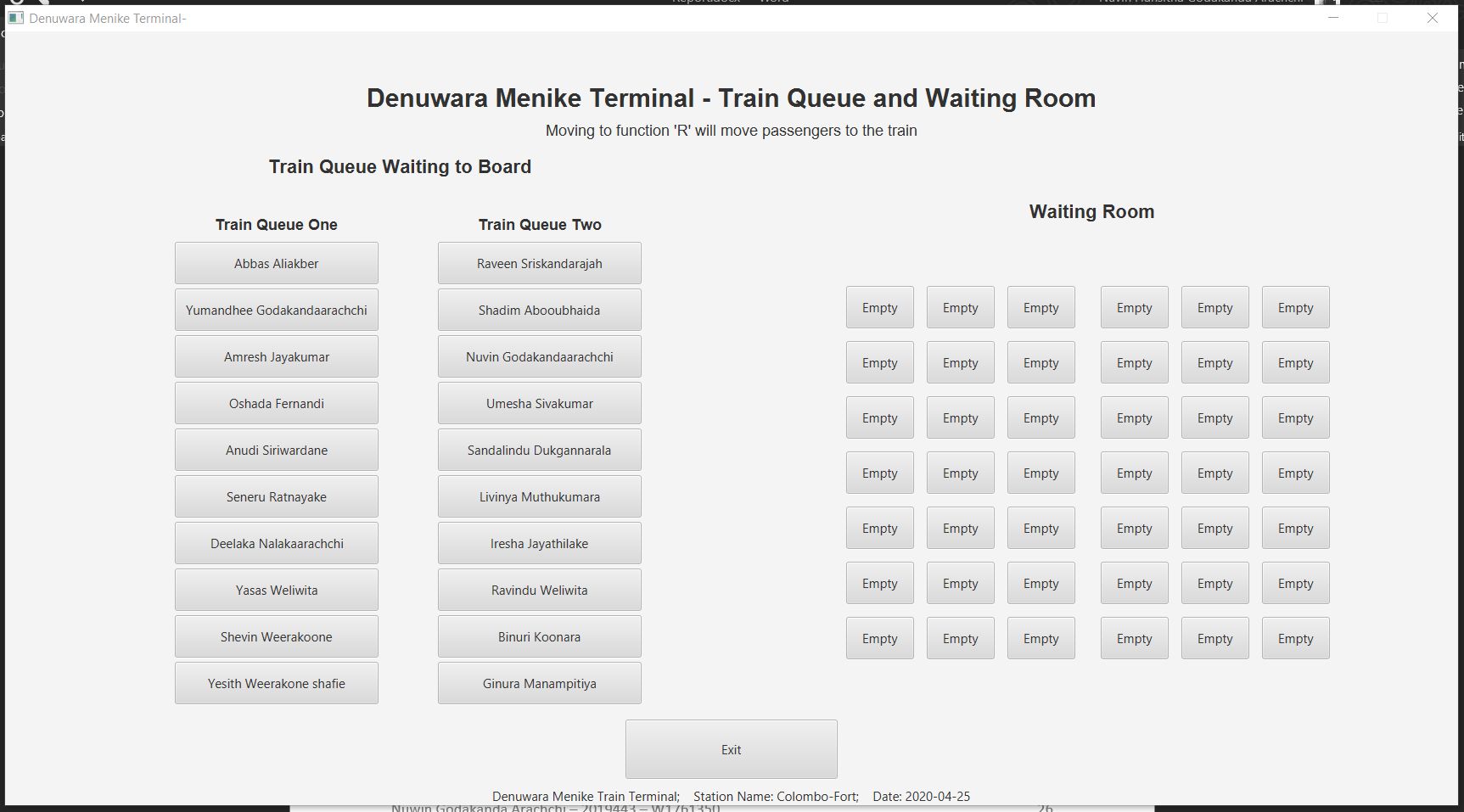
Output- Function Add Customers



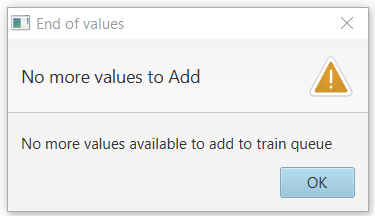
**Img 3.1**- Nominal giu presentation after pressng “a”



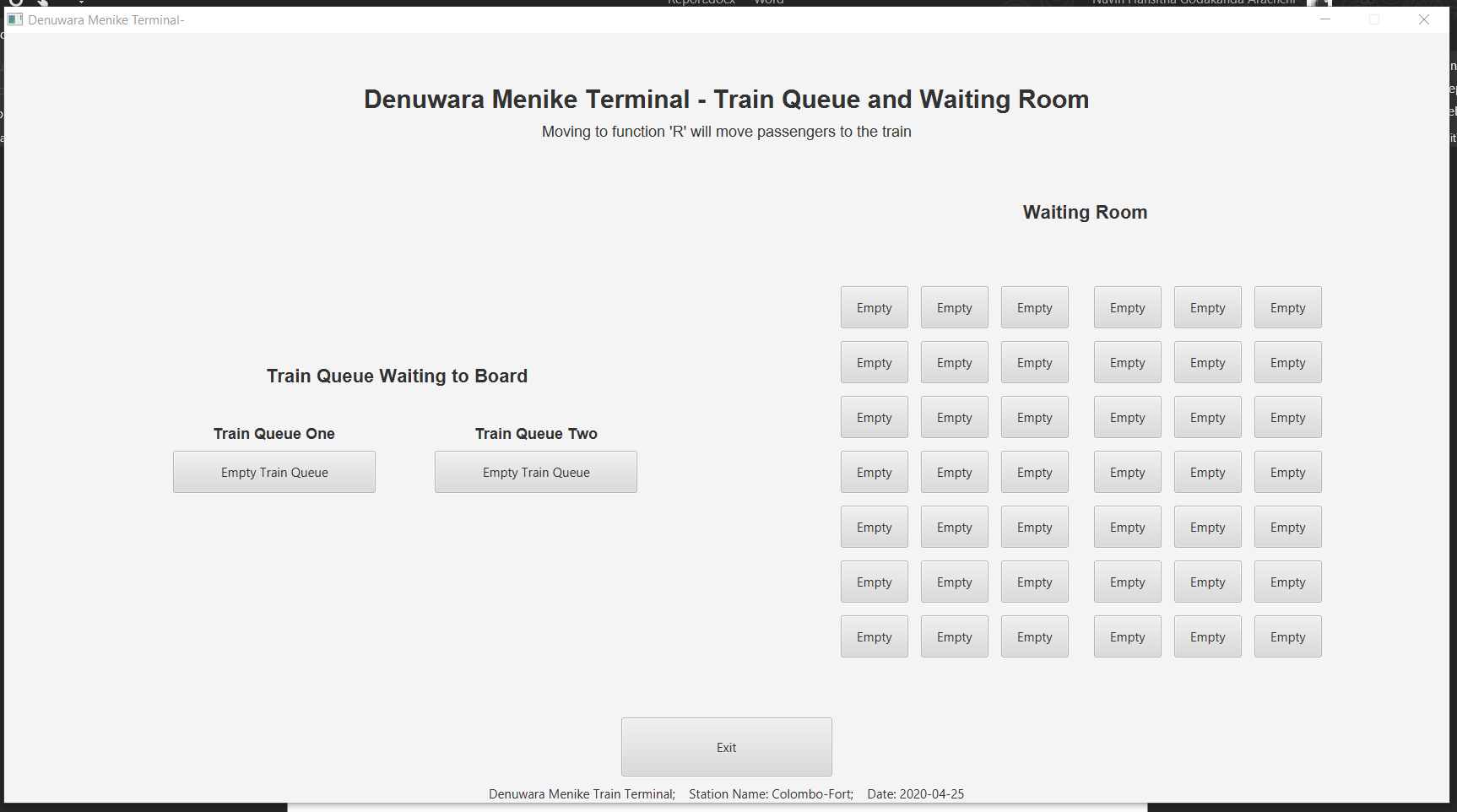
**Img 3.2-** After both train queues are full



**Img 3.4-** When train queue is full



**Img 3.6-** If no more seats are available to ass



**Img 3.8-** If no more seats are available to add

View the Train Queue – view()

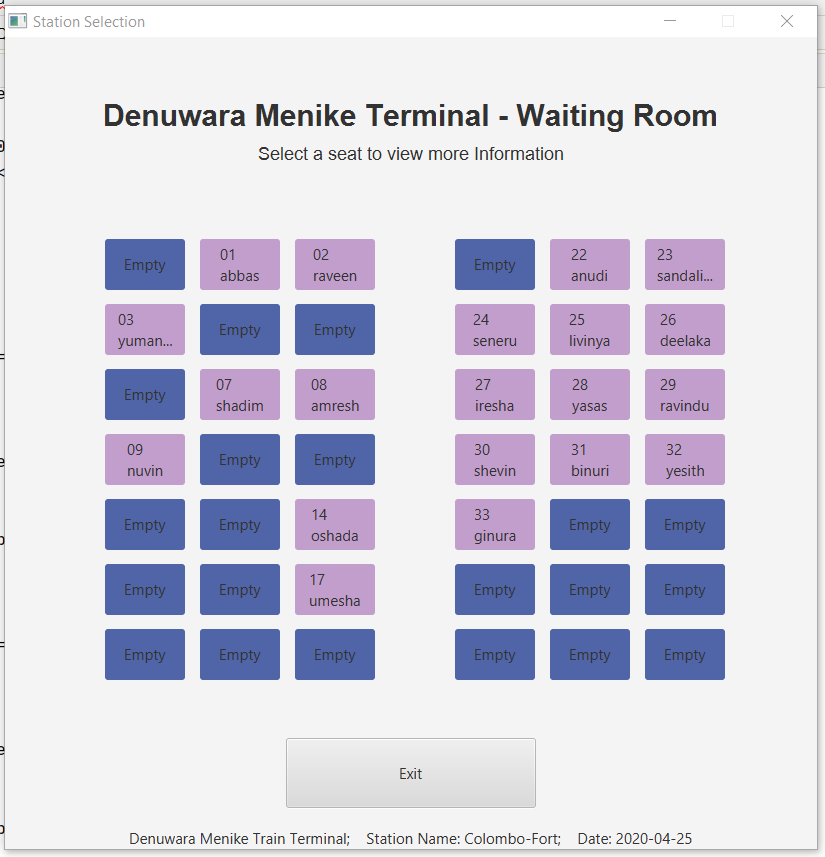
Function view train queue will allow the user to see all the seats in the train queue. The seat the passengers have booked will be shown in a different color, along with their name on it. To show more information of these passengers, once the seats are clicked, a message box will be popped with all the details about the customer.

Source Code- Function View Queue

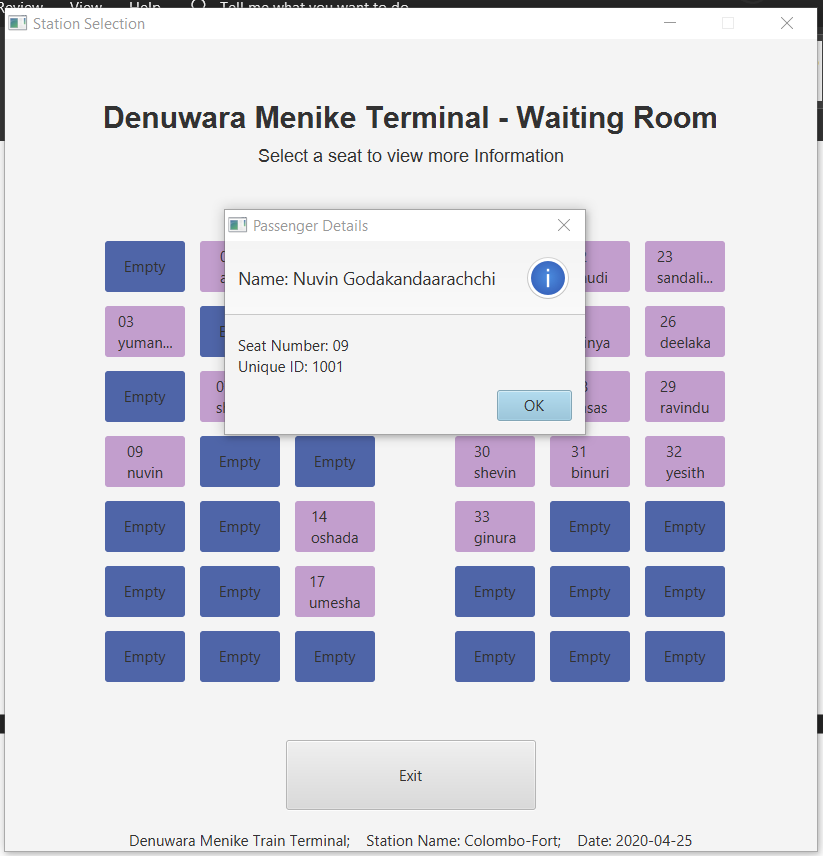
public static void view(){  
  
 //============================================================================================================//  
 // INITIALIZING ELEMENTS  
  
 //------------------------------------------------------------------------------------------------------- Stages  
  
 Stage stageOne = new Stage();  
 BorderPane rootOne = new BorderPane();  
 Scene sceneOne = new Scene(rootOne,800,800);  
 stageOne.setScene(sceneOne);  
 rootOne.getStylesheets().add("/style.css");  
 stageOne.setResizable(false);  
 stageOne.setTitle("Station Selection");  
 stageOne.show();  
  
 LocalDate date = LocalDate.now(); //local date  
  
 //------------------------------------------------------------------------------------------------------- Labels  
  
 Label mainLabel = new Label("Denuwara Menike Terminal - Waiting Room");  
 mainLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 30));  
 mainLabel.setPadding(new Insets(60,20,5,20));  
  
 Label subLabel = new Label("Select a seat to view more Information");  
 subLabel.setFont(Font.font("sans-serif", FontPosture.REGULAR, 18));  
 subLabel.setPadding(new Insets(5,20,5,20));  
  
 Label deets = new Label("Denuwara Menike Train Terminal; Station Name: "+stops[station]+  
 "; Date: "+date);  
  
  
 //------------------------------------------------------------------------------------------------------ Buttons  
  
 Button exit = new Button("Exit");  
 exit.setId("closeViewButton");  
  
  
 //============================================================================================================//  
 // GUI INITIALIZATION  
  
 //------------------------------------------------------------------------------------------Stage GUI Components  
  
 //main three boxes  
 VBox headerOne = new VBox();  
 HBox centerOne = new HBox(50);  
 VBox footerOne = new VBox(20);  
  
 //sub boxes  
 VBox centerLeft = new VBox(50);  
 VBox centerMid = new VBox(50);  
 VBox centerRight = new VBox();  
  
 //flowpanes for waiting room  
 FlowPane left = new FlowPane(15,15);  
 FlowPane mid = new FlowPane(15,15);  
  
 //positioning in border pane  
 rootOne.setTop(headerOne);  
 rootOne.setLeft(centerOne);  
 rootOne.setBottom(footerOne);  
  
 headerOne.setAlignment(Pos.CENTER);  
 centerOne.setAlignment(Pos.CENTER);  
 footerOne.setAlignment(Pos.CENTER);  
  
 //adding labels and buttons using getchildern  
 headerOne.getChildren().add(mainLabel);  
 headerOne.getChildren().add(subLabel);  
  
 centerOne.getChildren().add(centerLeft);  
 centerOne.getChildren().add(centerMid);  
 centerOne.getChildren().add(centerRight);  
  
 centerLeft.getChildren().add(left);  
 centerMid.getChildren().add(mid);  
  
 //------------ Waiting room gui program code  
 Button[] waitingButtons = new Button[42];  
  
 left.setPrefWrapLength(270); // setting a max width for box  
 left.setPadding(new Insets(70, 10, 10, 100));  
  
 for (int i=0;i<21;i++){  
 waitingButtons[i]=new Button("Empty"); //default buttons with text empty  
 waitingButtons[i].setId("waitingButtons");  
 waitingButtons[i].setStyle("-fx-background-color: #4f65a8");  
 left.getChildren().add(waitingButtons[i]); //added to root  
 }  
  
 mid.setPrefWrapLength(270);  
 mid.setPadding(new Insets(70, 10, 10, 20));  
 for (int i=21;i<42;i++){  
 waitingButtons[i]=new Button("Empty"); //default buttons with text empty  
 waitingButtons[i].setId("waitingButtons");  
 waitingButtons[i].setStyle("-fx-background-color: #4f65a8");  
 mid.getChildren().add(waitingButtons[i]); //added to root  
 }  
  
 for (int i=0;i<42;i++){  
 if (waitingRoom[i]!=null){ // if the waiting index has values  
 waitingButtons[Integer.parseInt(waitingRoom[i].getSeat())].setText(waitingRoom[i].getSeat()+"\n"+  
 waitingRoom[i].getName());  
 waitingButtons[Integer.parseInt(waitingRoom[i].getSeat())].setStyle("-fx-background-color: #c29ecd");  
 int finalI = i;  
 waitingButtons[i].setOnAction(event -> { // when a button is clicked  
 Alert info = new Alert(Alert.AlertType.INFORMATION);  
 info.setTitle("Passenger Details");  
 info.setHeaderText("Name: "+waitingRoom[finalI].getFullName());  
 info.setContentText("Seat Number: "+waitingRoom[finalI].getSeat()+"\n"+  
 "Unique ID: "+waitingRoom[finalI].getId());  
 info.showAndWait();  
 });  
 }  
 }  
  
 footerOne.getChildren().add(exit);  
 footerOne.getChildren().add(deets);  
  
 exit.setOnAction(event -> {  
 stageOne.close();  
 menu();  
 });  
  
}

**Additional functions used from class Passenger.java and PassengerQueue.java have been used.**

Output- Function View Queue



**Img 4.1-** V pressed at any point



**Img 4.2-** After selecting a passenger;

Delete Passenger from train queue method – deleteCustomer()

The delete function allows the user to delete himself either from the train queue to the waiting room, or to leave the train station straight off. The program will firstly prompt the user to select if he wants to go back to the waiting room or to leave the train station. If the program detects an erroneous input, they will handle it accordingly. This proves the validation and verification used in the program to account for human errors.

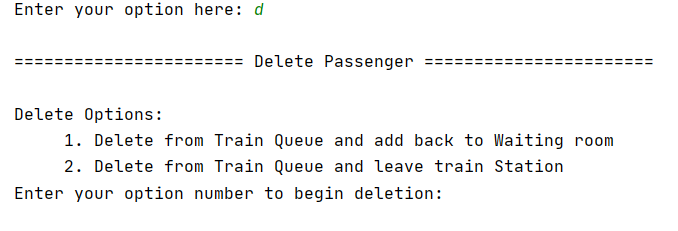
Source Code- Function Delete Customer

public static void delete(){  
  
 Scanner input = new Scanner(System.in);  
 System.out.println("Delete Options: ");  
 System.out.println(" 1. Delete from Train Queue and add back to Waiting room");  
 System.out.println(" 2. Delete from Train Queue and leave train Station");  
 System.out.println("Enter your option number to begin deletion: ");  
 String deleteType = input.next(); // checking if passenger wants to leave the train station or just the queue  
 switch (deleteType){  
 case "1": //if the selection is one  
 System.out.println("Enter your first name or Unique ID to delete: ");  
 String name= input.next().toLowerCase();// name will be made to lowercase for verification  
 boolean found= trainQueueOne.delete(name); // checks for name in delete fucntion in passenger queue class  
 boolean nextFound;  
 if (!found){ // if boolean returned from queue one search is false  
 nextFound= trainQueueTwo.delete(name); //check in other queue  
 if (!nextFound){// if still failed t find, propmts no name found  
 System.out.println("Oops! Couldn't find that! Want to give it another try? ");  
 System.out.println("Press <O> to try again, or any other key to move back to the main menu: ");  
 String choice= input.next().toLowerCase();  
 if (choice.equals("o")){ // gives a chance for the user to start over  
 delete();  
 }else{  
 System.out.println("Process Exit. No changes to local.");  
 menu(); //calling back menu to sustain program  
 }  
 }else{  
 System.out.println("Process Exit. Local updated. Make sure to press upload the data.");  
 menu(); //calling back menu to sustain program- On press of exit button  
 }  
 }  
 break;  
 case "2":  
 System.out.println("Enter your first name or Unique ID to delete: ");  
 name= input.next().toLowerCase();// name will be made to lowercase for verification  
 found= trainQueueOne.delete(name); // checks for name in delete fucntion in passenger queue class  
 if (!found){ // if boolean returned from queue one search is false  
 nextFound= trainQueueTwo.delete(name); //check in other queue  
 if (!nextFound){// if still failed t find, propmts no name found  
 System.out.println("Oops! Couldn't find that! Want to give it another try? ");  
 System.out.println("Press <O> to try again, or any other key to move back to the main menu: ");  
 String choice= input.next().toLowerCase();  
 if (choice.equals("o")){ // gives a chance for the user to start over  
 delete();  
 }else{  
 System.out.println("Process Exit. No changes to local.");  
 menu(); //calling back menu to sustain program  
 break;  
 }  
 }else{  
 System.out.println("Process Exit. Local updated. Make sure to press upload the data.");  
 menu(); //calling back menu to sustain program- On press of exit button  
 break;  
 }  
 }  
 for (int i=0;i<42;i++){ // removes passenger from waiting room  
 if (waitingRoom[i] != null && (waitingRoom[i].getName().contains(name) ||  
 (waitingRoom[i].getId().equals(name)))) {  
 waitingRoom[i] = null; // removing as in making it null  
 break;  
 }  
 }  
 menu();  
 break;  
 default: // default will call back the function  
 System.out.println("Oops! That's an incorrect input!");  
 delete();  
 }  
}

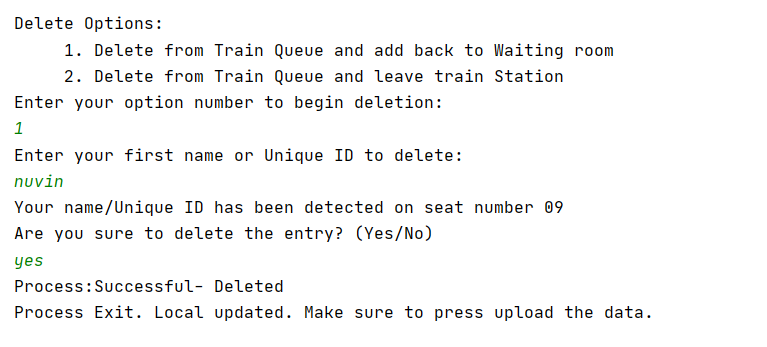
**Additional delete method used for deletion in PassengerQueue.java**

public boolean delete(String name){  
 Scanner input = new Scanner(System.in);  
 for (int traverse=0;traverse<10;traverse++){  
 if (queueArray[traverse]!=null) {  
 if (queueArray[traverse].getName().toLowerCase().contains(name) || queueArray[traverse].getId().contains(name)) {  
 System.out.println("Your name/Unique ID has been detected on seat number "+queueArray[traverse].getSeat());  
 System.out.println("Are you sure to delete the entry? (Yes/No)");  
 String choice = input.next().toLowerCase();  
 switch (choice){  
 case "yes":  
 System.out.println("Process:Successful- Deleted");  
 queueArray[traverse].setAdded(false);  
 queueArray[traverse]=null;  
 for (int numOne = traverse; numOne < 9; numOne++) { // Using classical bubble sort  
 if (queueArray[numOne+1]!=null) {  
 queueArray[numOne] = queueArray[numOne + 1];  
 queueArray[numOne + 1]=null;  
 }  
 }  
 maxLength--;  
 last--;  
 return true;  
 case "no":  
 System.out.println("Process:Terminated- Not Deleted");  
 return true;  
 }  
 }  
 }  
 }  
 return false;  
}

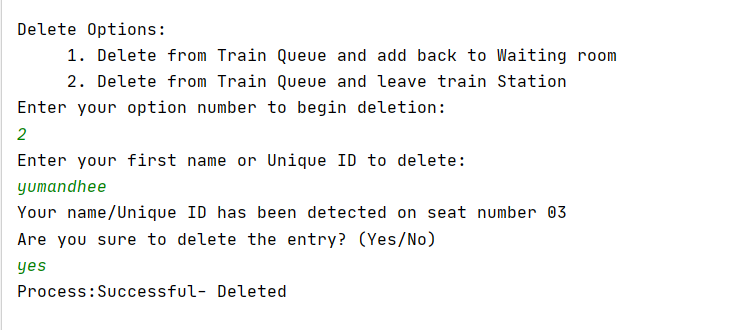
Output- Function Delete Customer



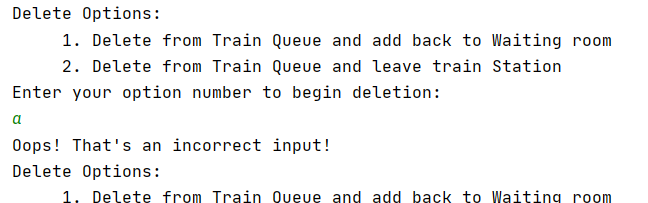
**Img 5.1-** As user enters “D” console will propmt to get user choice



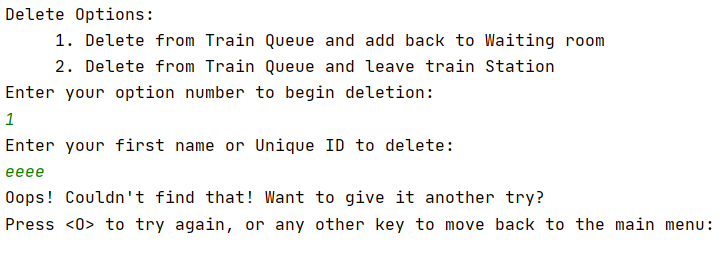
**Img 5.2-** Normal delete procedure



**Img 5.3-** Normal delete Procedure 2



**Img 5.4-** Incorrect input for input 1



**Img 5.5-** If name is not found, the user will be prompted to chose to go to the menu or retry delete

Store data to plain Text File – store()

This function with allow the program to directly store its data into a plain text file, serialized. The data is serialized to allow the data to read back to the data structure.

Source Code- Function Store

public static void store(){  
 try{  
 System.out.println("Store file Opened");  
 File storeQueue = new File("queueDetails.txt"); //opens file  
 FileOutputStream fileOutputStream = new FileOutputStream(storeQueue); //opens file for write  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream); //passes for objects  
 System.out.println("Store file Opened");  
  
 // values passed in  
 objectOutputStream.writeObject(station);  
 objectOutputStream.writeObject(direction);  
 objectOutputStream.writeObject(waitingRoom);  
 objectOutputStream.writeObject(trainQueueOne.getQueue());  
 objectOutputStream.writeObject(trainQueueOne.storeAdditionalAll());  
 objectOutputStream.writeObject(trainQueueTwo.getQueue());  
 objectOutputStream.writeObject(trainQueueTwo.storeAdditionalAll());  
 objectOutputStream.writeObject(boardedToTrain);  
  
 objectOutputStream.flush();  
 System.out.println("Store Successful");  
 objectOutputStream.close();  
 System.out.println("Store Closed: Success");  
 fileOutputStream.close();  
 } catch (IOException e) {  
 System.out.println("Error loading data to file!");  
 }  
 menu();  
}

Retreive data from text file – load()

This function allows the program to read all the data it stored into the text file, back from it. Since the code uses serialized codes, the retrieval won’t have an issue.

Source-Code- load method

public static void load(){  
 try{  
 System.out.println("Load file Opened");  
 File storeQueue = new File("queueDetails.txt");  
 FileInputStream fileInputStream = new FileInputStream(storeQueue);  
 ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);  
  
 station= (int) objectInputStream.readObject();  
 direction= (String) objectInputStream.readObject();  
 waitingRoom=((Passenger[]) objectInputStream.readObject());  
 trainQueueOne.setQueue((Passenger[]) objectInputStream.readObject());  
 trainQueueOne.initialize((int[]) objectInputStream.readObject());  
 trainQueueTwo.setQueue((Passenger[]) objectInputStream.readObject());  
 trainQueueTwo.initialize((int[]) objectInputStream.readObject());  
 boardedToTrain= (ArrayList<Passenger>) objectInputStream.readObject();  
 System.out.println("Load Complete");  
  
 System.out.println("Load Closed: Success");  
  
 }catch (IOException | ClassNotFoundException e) {  
 System.out.println("Error retrieving data from file");  
 }  
 menu();  
}

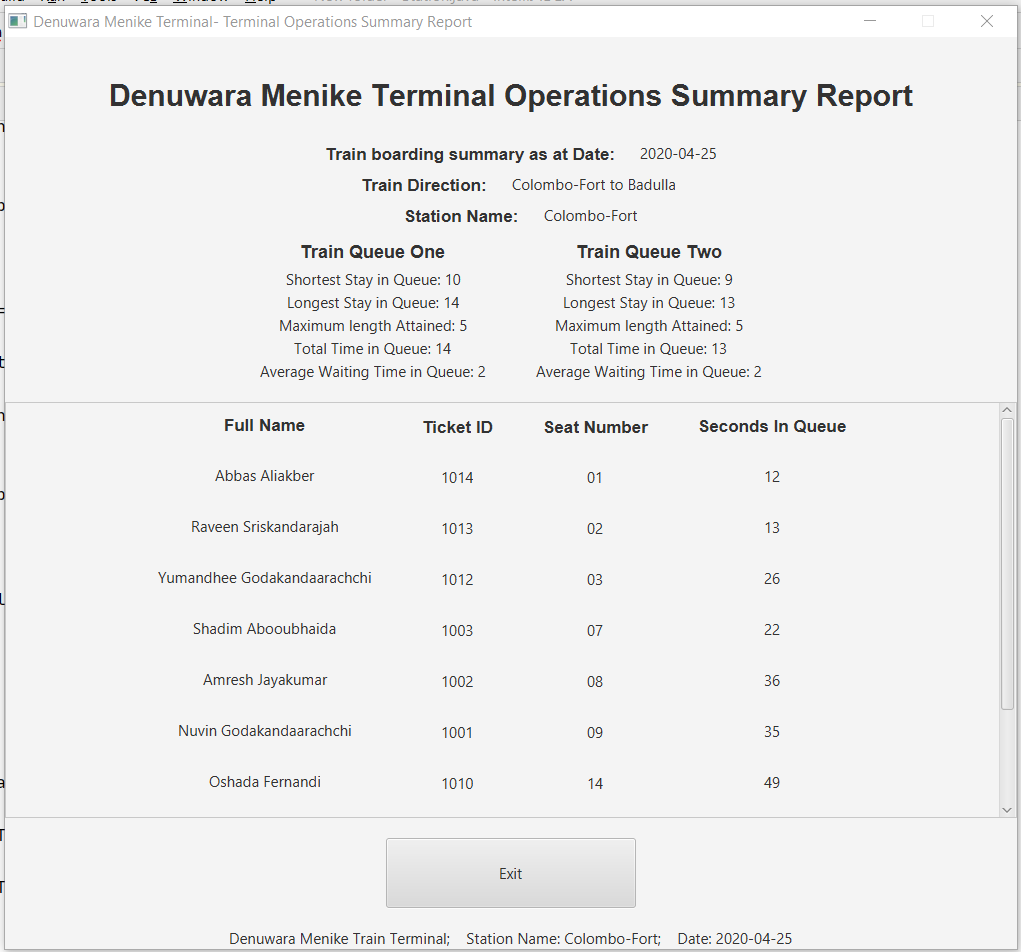
Run simulation method – run()

Method run simulation will remove all the current passengers from the train queue and all the passengers to the train (i.e. board to the train). While this happens, the passenger queue class, explained below, will collect all the information needed about the queue. This will be used to show a summary of details of each queue used.

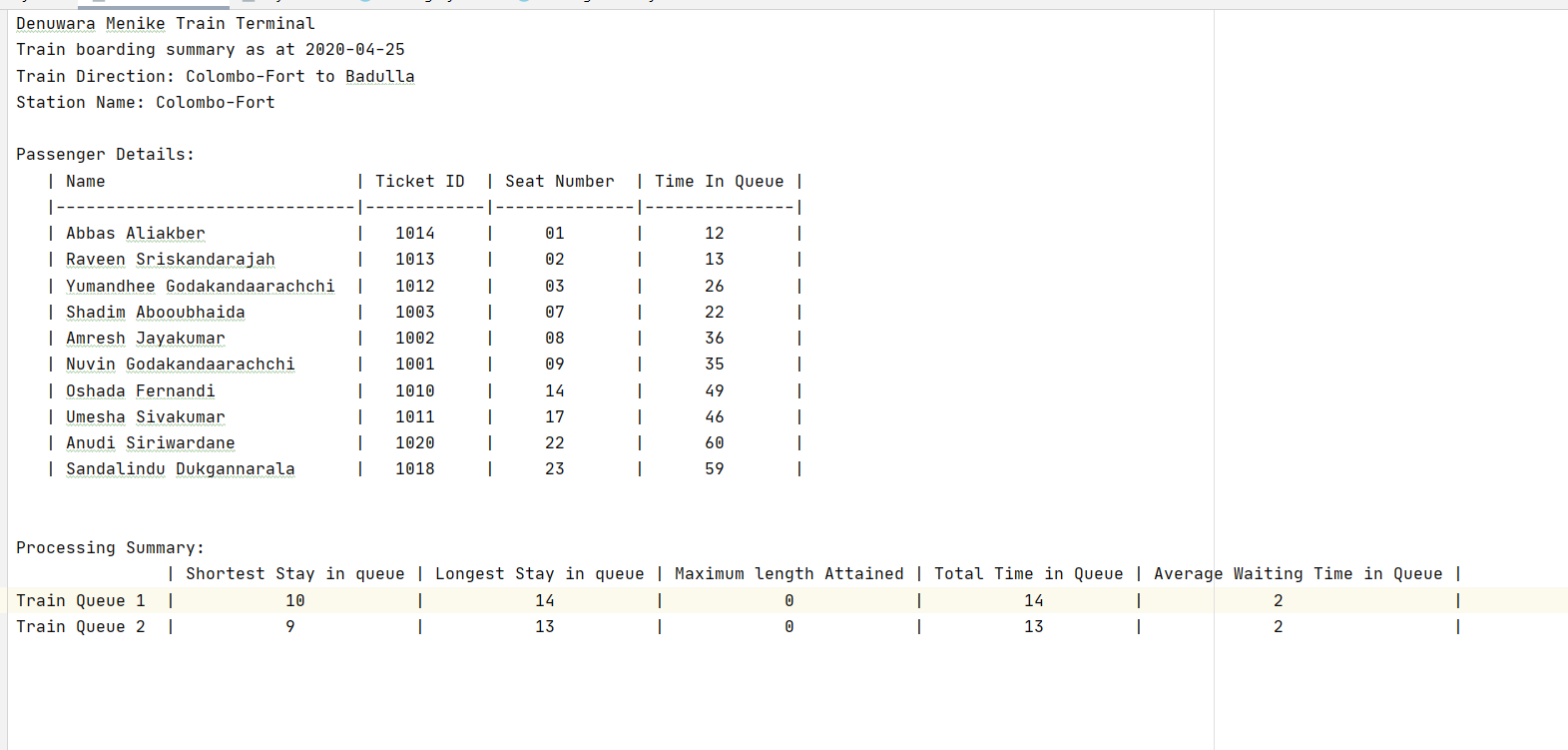
Source-Code -run method

public static void run () {  
  
 // adding to train from train queue - ie. boarding  
 boolean repeat=false;  
 if (trainQueueOne.getLength()!=0) {  
 do {  
 Passenger passenger = trainQueueOne.remove();  
 boardedToTrain.add(passenger);  
 System.out.println(passenger.getFullName() + " added to Train");  
 if (trainQueueOne.getLength() == 0) {  
 repeat = true;  
 }  
 } while (!repeat);  
 }  
  
 // adding to train from train queue - ie. boarding  
 repeat=false;  
 if (trainQueueTwo.getLength()!=0) {  
 do {  
 Passenger passenger = trainQueueTwo.remove();  
 boardedToTrain.add(passenger);  
 System.out.println(passenger.getFullName() + " added to Train");  
 if (trainQueueTwo.getLength() == 0) {  
 repeat = true;  
 }  
 } while (!repeat);  
 }  
  
 //============================================================================================================//  
 // INITIALIZING ELEMENTS  
  
 //------------------------------------------------------------------------------------------------------- Stages  
  
 Stage stage = new Stage();  
 BorderPane root = new BorderPane();  
 Scene scene = new Scene(root,1000,900);  
 stage.setScene(scene);  
 root.getStylesheets().add("/style.css");  
 stage.setResizable(false);  
 stage.setTitle("Denuwara Menike Terminal- Terminal Operations Summary Report");  
 stage.show();  
  
 LocalDate date = LocalDate.now(); //local date  
  
  
 //------------------------------------------------------------------------------------------------------- Labels  
  
 Label mainLabel = new Label("Denuwara Menike Terminal Operations Summary Report");  
 mainLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 30));  
 mainLabel.setPadding(new Insets(40,20,20,20));  
  
 Label deets = new Label("Denuwara Menike Train Terminal; Station Name: "+stops[station]+  
 "; Date: "+date);  
  
 Label dateInfo = new Label("Train boarding summary as at Date: ");  
 dateInfo.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
 dateInfo.setPadding(new Insets(0,20,0,20));  
  
 Label nowDate = new Label(date.toString());  
  
 Label directionLabel = new Label("Train Direction: ");  
 directionLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
 directionLabel.setPadding(new Insets(0,20,0,20));  
  
 Label side = new Label();  
 if (direction.equals("ctb")) {  
 side.setText("Colombo-Fort to Badulla ");  
 }else{  
 side.setText("Badulla to Colombo-Fort ");  
 }  
  
 Label stationLabel = new Label("Station Name: ");  
 stationLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
 stationLabel.setPadding(new Insets(0,20,0,20));  
  
 Label stationInfo = new Label(stops[station]);  
  
 Label queueOneLabel = new Label("Train Queue One");  
 queueOneLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,18));  
 queueOneLabel.setPadding(new Insets(0,0,5,0));  
  
 Label queueTwoLabel = new Label("Train Queue Two");  
 queueTwoLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,18));  
 queueTwoLabel.setPadding(new Insets(0,0,5,0));  
  
 //adding data from train queues to labels  
  
 Label queueOneShortest = new Label("Shortest Stay in Queue: "+trainQueueOne.getShortestStay());  
  
 Label queueOneLongest = new Label("Longest Stay in Queue: "+trainQueueOne.getLongestStay());  
  
 Label queueOneLength = new Label("Maximum length Attained: "+trainQueueOne.getLongestLength());  
  
 Label queueOneMaxStay = new Label("Total Time in Queue: "+trainQueueOne.getLongestStay());  
  
 Label queueOneAverage = new Label("Average Waiting Time in Queue: "+trainQueueOne.getAverage());  
  
  
 Label queueTwoShortest = new Label("Shortest Stay in Queue: "+trainQueueTwo.getShortestStay());  
  
 Label queueTwoLongest = new Label("Longest Stay in Queue: "+trainQueueTwo.getLongestStay());  
  
 Label queueTwoLength = new Label("Maximum length Attained: "+trainQueueTwo.getLongestLength());  
  
 Label queueTwoMaxStay = new Label("Total Time in Queue: "+trainQueueTwo.getLongestStay());  
  
 Label queueTwoAverage = new Label("Average Waiting Time in Queue: "+trainQueueTwo.getAverage());  
  
 Label fullName = new Label ("Full Name");  
 fullName.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
  
 Label id = new Label("Ticket ID");  
 id.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
  
 Label seatNumber = new Label("Seat Number");  
 seatNumber.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
  
 Label secondsLabel = new Label("Seconds In Queue");  
 secondsLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
  
  
 //------------------------------------------------------------------------------------------------------ Buttons  
  
 Button exit = new Button("Exit");  
 exit.setId("closeRun");  
  
 //============================================================================================================//  
 // GUI PROGRAM CODES  
  
 //------------------------------------------------------------------------------------------------------- Stages  
  
 VBox header = new VBox(10); //main boxes  
 HBox center = new HBox(50);  
 ScrollPane centerScroll = new ScrollPane(center);  
  
 HBox headerSummary = new HBox(50); //used to show summary of queues  
  
 HBox dateBox = new HBox(); //used to show important aspects of the data  
 HBox directionBox = new HBox();  
 HBox stationBox = new HBox();  
  
 VBox queueOne = new VBox(2); //data of each queue  
 queueOne.setPadding(new Insets(5,0,20,0));  
 VBox queueTwo = new VBox(2);  
 queueTwo.setPadding(new Insets(5,0,20,0));  
  
 VBox names = new VBox(30);  
 VBox ids = new VBox(30);  
 VBox seat = new VBox(30);  
 VBox seconds = new VBox(30);  
  
 VBox footer = new VBox(20);  
 footer.setPadding(new Insets(20,0,0,0));  
  
 names.setPadding(new Insets(9,0,0,0));  
 ids.setPadding(new Insets(14,0,0,0));  
 seat.setPadding(new Insets(14,0,0,0));  
 seconds.setPadding(new Insets(11,0,0,0));  
  
 root.setTop(header);  
 root.setCenter(centerScroll);  
 root.setBottom(footer);  
  
 // adding children to get  
 header.setAlignment(Pos.CENTER);  
 header.getChildren().add(mainLabel);  
 header.getChildren().add(dateBox);  
 header.getChildren().add(directionBox);  
 header.getChildren().add(stationBox);  
 header.getChildren().add(headerSummary);  
  
 dateBox.setAlignment(Pos.CENTER);  
 dateBox.getChildren().add(dateInfo);  
 dateBox.getChildren().add(nowDate);  
  
 directionBox.setAlignment(Pos.CENTER);  
 directionBox.getChildren().add(directionLabel);  
 directionBox.getChildren().add(side);  
  
 stationBox.setAlignment(Pos.CENTER);  
 stationBox.getChildren().add(stationLabel);  
 stationBox.getChildren().add(stationInfo);  
  
 headerSummary.setAlignment(Pos.CENTER);  
 headerSummary.getChildren().add(queueOne);  
 headerSummary.getChildren().add(queueTwo);  
  
 // ----------------------adding the important data  
 // adding information of train queue one  
 queueOne.setAlignment(Pos.CENTER);  
 queueOne.getChildren().add(queueOneLabel);  
 queueOne.getChildren().add(queueOneShortest);  
 queueOne.getChildren().add(queueOneLongest);  
 queueOne.getChildren().add(queueOneLength);  
 queueOne.getChildren().add(queueOneMaxStay);  
 queueOne.getChildren().add(queueOneAverage);  
  
 //adding information of train queue two  
 queueTwo.setAlignment(Pos.CENTER);  
 queueTwo.getChildren().add(queueTwoLabel);  
 queueTwo.getChildren().add(queueTwoShortest);  
 queueTwo.getChildren().add(queueTwoLongest);  
 queueTwo.getChildren().add(queueTwoLength);  
 queueTwo.getChildren().add(queueTwoMaxStay);  
 queueTwo.getChildren().add(queueTwoAverage);  
  
 //adding labels for name, id. seat, seconds  
 center.setAlignment(Pos.CENTER);  
 center.getChildren().add(names);  
 center.getChildren().add(ids);  
 center.getChildren().add(seat);  
 center.getChildren().add(seconds);  
 centerScroll.setContent(center);  
  
 names.setAlignment(Pos.CENTER);  
 ids.setAlignment(Pos.CENTER);  
 seat.setAlignment(Pos.CENTER);  
 seconds.setAlignment(Pos.CENTER);  
 centerScroll.setFitToWidth(true);  
  
 names.getChildren().add(fullName);  
 ids.getChildren().add(id);  
 seat.getChildren().add(seatNumber);  
 seconds.getChildren().add(secondsLabel);  
  
 // this for loop will add values to the boxes  
 for (Passenger passenger : waitingRoom) {  
 if (passenger != null && passenger.getAdded()) {  
 names.getChildren().add(new Label (passenger.getFullName()));// adds to name box  
 ids.getChildren().add(new Label (passenger.getId())); // adds to id box  
 seat.getChildren().add(new Label(passenger.getSeat())); // adds to seat box  
 seconds.getChildren().add(new Label(String.valueOf(passenger.getSeconds()))); // adds to seconds box  
 }  
 }  
  
 footer.setAlignment(Pos.CENTER);  
 footer.getChildren().add(exit);  
 exit.setOnAction(event -> { //exit button soa  
 stage.close();  
 menu();  
 });  
 footer.getChildren().add(deets);  
  
 System.out.println(seatList.size());  
  
 try {  
 int longest=0;  
 for (int i=0;i<42;i++){  
 //Comparing sizes of the names to get longest  
 if (waitingRoom[i]!=null && waitingRoom[i].getFullName().length()>longest) {  
 longest=waitingRoom[i].getFullName().length();  
 }  
 }  
  
 //========================================================================================================//  
 // STORE TO FILE  
  
 // Format has been made according to eye patterns.  
 File storeQueue = new File("Report\_on\_"+date+".txt");  
 FileWriter writer = new FileWriter(storeQueue);  
 writer.write("Denuwara Menike Train Terminal");  
 writer.write("\r\n");  
 writer.write("Train boarding summary as at " + date);  
 writer.write("\r\n");  
 if (direction.equals("ctb")) {  
 writer.write("Train Direction: Colombo-Fort to Badulla ");  
 }else{  
 writer.write("Train Direction: Badulla to Colombo-Fort ");  
 }  
 writer.write("\r\n");  
 writer.write("Station Name: "+stops[station]);  
 writer.write("\r\n");writer.write("\r\n");  
 writer.write("Passenger Details: ");  
 writer.write("\r\n");  
 writer.write(" | Name");  
 for (int i=0;i<=longest-4;i++){ writer.write(" "); }  
 writer.write(" | Ticket ID | Seat Number | Time In Queue |");  
 writer.write("\r\n"); // Move to next line  
 writer.write(" |-----");  
 for (int i=0;i<=longest-4;i++){ writer.write("-"); }  
 writer.write("-|------------|--------------|---------------| ");  
 writer.write("\r\n");  
 for (Passenger passenger : waitingRoom) {  
 if (passenger != null && passenger.getAdded()) {  
 writer.write(" | ");  
 writer.write(passenger.getFullName());  
 for (int nextI = 0; nextI <= longest - passenger.getFullName().length(); nextI++) {  
 writer.write(" ");  
 }  
 writer.write(" | " + passenger.getId() + " ");  
 writer.write(" | " + passenger.getSeat() + " ");  
 if (passenger.getSeconds()<10){  
 writer.write(" | 0" + passenger.getSeconds() + " |");  
 }else{  
 writer.write(" | " + passenger.getSeconds() + " |");  
 }  
 writer.write("\r\n");  
 }  
 }  
 writer.write("\r\n");writer.write("\r\n");  
 writer.write("Processing Summary: ");  
 writer.write("\r\n");  
 writer.write(" | Shortest Stay in queue | Longest Stay in queue |" +  
 " Maximum length Attained | Total Time in Queue | Average Waiting Time in Queue |");  
 writer.write("\r\n");  
 writer.write("Train Queue 1 | "+trainQueueOne.getShortestStay()+ " | "+  
 trainQueueOne.getLongestStay()+ " | "+trainQueueOne.getLength()+  
 " | "+trainQueueOne.getMaxStay()+ " | "+  
 trainQueueOne.getAverage()+" |");  
 writer.write("\r\n");  
 writer.write("Train Queue 2 | "+trainQueueTwo.getShortestStay()+ " | "+  
 trainQueueTwo.getLongestStay()+ " | "+trainQueueTwo.getLength()+  
 " | "+trainQueueTwo.getMaxStay()+ " | "+  
 trainQueueTwo.getAverage()+" |");  
 writer.close();  
 }catch (IOException e) {  
 System.out.println("Error loading data to file.");  
 }  
}

Output- run method



**Img 8.1-** Report produced in gui using values from both queues, and personal data



**Img 8.2-** The text file that will be output. Pretty printed.

Class Passenger Queue- PassengerQueue()

package sample;  
  
import java.util.Scanner;  
  
public class PassengerQueue {  
  
 private Passenger[] queueArray = new Passenger[10];  
 private int first;  
 private int last;  
 private int maxStayInQueue;  
 private int maxLength;  
 private int shortestStay=1000;  
 private int longestStay=0;  
 private int totalTime;  
 private int longestLength;  
  
 public void add (Passenger next){ //Setter  
 if (isFull()){  
 System.out.println("Train queue is full");  
 }else{  
 for (int numOne = 0; numOne < maxLength-1; numOne++) { // Using classical bubble sort  
 for (int numTwo = numOne + 1; numTwo < maxLength-1; numTwo++) {  
 // comparing adjacent strings  
 int one= Integer.parseInt(queueArray[numOne].getSeat());  
 int two= Integer.parseInt(queueArray[numTwo].getSeat());  
 if (two<one) {  
 Passenger cache = queueArray[numTwo]; // sorting the seats according to  
 queueArray[numTwo] = queueArray[numOne];  
 queueArray[numOne] = cache;  
 }  
 }  
 }  
 queueArray[last] = next;  
 last++;  
 maxLength++;  
 longestLength++;  
 }  
 }  
  
 public Passenger remove(){ //Setter  
 Passenger current = queueArray[first];  
 if (isEmpty()) {  
 System.out.println("Train queue is empty");  
 }else{  
 for (int numOne = 0; numOne < 9; numOne++) { // Using classical bubble sort  
 if (queueArray[numOne+1]!=null) {  
 queueArray[numOne] = queueArray[numOne + 1];  
 queueArray[numOne + 1]=null; // passing the null value to the next available slot  
 }else{  
 queueArray[numOne]=null;  
 }  
 }  
 maxLength--;  
 last--;  
 }  
 return current;  
 }  
  
 public String accessName(int index){  
 return queueArray[index].getFullName(); // accessing the full name through the queue array  
 }  
  
 public boolean delete(String name){  
 Scanner input = new Scanner(System.in);  
 for (int traverse=0;traverse<10;traverse++){  
 if (queueArray[traverse]!=null) {  
 if (queueArray[traverse].getName().toLowerCase().contains(name) || queueArray[traverse].getId().contains(name)) {  
 System.out.println("Your name/Unique ID has been detected on seat number "+queueArray[traverse].getSeat());  
 System.out.println("Are you sure to delete the entry? (Yes/No)");  
 String choice = input.next().toLowerCase();  
 switch (choice){  
 case "yes":  
 System.out.println("Process:Successful- Deleted");  
 queueArray[traverse].setAdded(false);  
 queueArray[traverse]=null;  
 for (int numOne = traverse; numOne < 9; numOne++) { // Using classical bubble sort  
 if (queueArray[numOne+1]!=null) {  
 queueArray[numOne] = queueArray[numOne + 1];  
 queueArray[numOne + 1]=null;  
 }  
 }  
 maxLength--;  
 last--;  
 return true;  
 case "no":  
 System.out.println("Process:Terminated- Not Deleted");  
 return true;  
 }  
 }  
 }  
 }  
 return false;  
 }  
  
 public boolean isEmpty(){ //Getter  
 return last == first;  
 }  
  
 public boolean isFull(){ //Getter  
 return maxLength == 10;  
 }  
  
 public void display(){ //Getter  
 if (isEmpty()) {  
 System.out.println("Empty Train Queue.");  
 }else{  
 System.out.println("Items: ");  
 for (int i=0;i<maxLength;i++){  
 System.out.println(queueArray[i].getName());  
 }  
 }  
 System.out.println(" ");  
 }  
  
 public int getLength(){  
 return maxLength;  
 }  
  
 public int getMaxStay(){ //Getter  
 return totalTime;  
 }  
  
 public Object getQueue(){  
 return queueArray;  
 }  
  
 public void setQueue(Passenger[] temp){  
 queueArray = temp;  
 int count=0;  
 for (Passenger passenger : queueArray) {  
 if (passenger != null) {  
 count++;  
 }  
 }  
 maxLength=count;  
 last=maxLength-1;  
 }  
  
 public void setTime(int time){  
 maxStayInQueue+=time;  
 queueArray[last-1].setSecondsInQueue(maxStayInQueue);  
  
 if (time>longestStay){  
 longestStay=time;  
 }  
 if (time<shortestStay){  
 shortestStay=time;  
 }  
 }  
  
 public int getShortestStay(){  
 return shortestStay;  
 }  
  
 public int getLongestStay(){  
 totalTime=longestStay;  
 return longestStay;  
 }  
  
 public int getLongestLength(){  
 return longestLength;  
 }  
  
 public int getAverage(){  
 return longestStay/longestLength;  
 }  
  
 public Object storeAdditionalAll(){  
 int[] essentials = new int[8];  
 essentials[0]=first;  
 essentials[1]=last;  
 essentials[2]=maxStayInQueue;  
 essentials[3]=maxLength;  
 essentials[4]=shortestStay;  
 essentials[5]=longestStay;  
 essentials[6]=totalTime;  
 essentials[7]=longestLength;  
 return essentials;  
 }  
  
 public void initialize(int[] essentials){  
 first=essentials[0];  
 last=essentials[1];  
 maxStayInQueue=essentials[2];  
 maxLength=essentials[3];  
 shortestStay=essentials[4];  
 longestStay=essentials[5];  
 totalTime=essentials[6];  
 longestLength=essentials[7];  
 }

Class Passenger- Passenger()

package sample;  
  
import java.io.Serializable;  
  
public class Passenger implements Serializable {  
  
 private String firstName;  
 private String surname;  
 private String uniqueId;  
 private String seatNumber;  
 private boolean addedQueue;  
 private int secondsInQueue=1;  
  
 public Passenger(String firstName, String surname, String uniqueId, String seatNumber) {  
 super();  
 this.uniqueId=uniqueId;  
 this.seatNumber=seatNumber;  
 setName(firstName, surname);  
 }  
  
  
 public String getName() {  
 return firstName;  
 }  
  
 public String getId(){  
 return uniqueId;  
 }  
  
 public String getSeat(){  
 return seatNumber;  
 }  
  
 public String getFullName(){  
 firstName=firstName.substring(0,1).toUpperCase()+firstName.substring(1);  
 surname=surname.substring(0,1).toUpperCase()+surname.substring(1);  
 return firstName+" "+surname;  
 }  
  
 public void setName(String name, String secondName) {  
 this.firstName=name;  
 this.surname=secondName;  
 }  
  
 public void setAdded(boolean addedQueue){  
 this.addedQueue=addedQueue;  
 }  
  
 public boolean getAdded(){  
 return addedQueue;  
 }  
  
 public int getSeconds() {  
 return secondsInQueue;  
 }  
  
 public void setSecondsInQueue(int sec) {  
 this.secondsInQueue=sec;  
 }  
  
 public void display() {  
  
 }  
  
}

Css Style

#combo{  
 -fx-min-width: 250;  
}  
#stationButtons{  
 -fx-min-width: 250;  
 -fx-min-height:70;  
}  
#addToQueue{  
 -fx-min-width: 250;  
 -fx-min-height:70;  
}  
#checkInButtons{  
 -fx-min-width: 250;  
}  
#closeCheckIn{  
 -fx-min-width: 250;  
 -fx-min-height:70;  
}  
#waitingButtons{  
 -fx-min-width: 80;  
 -fx-min-height: 50;  
 -fx-max-width: 80;  
 -fx-max-height: 50;  
}  
#seatButtons{  
 -fx-max-width: 100;  
 -fx-max-height: 100;  
}  
#closeViewButton{  
 -fx-min-width: 250;  
 -fx-min-height:70;  
}  
#closeRun{  
 -fx-min-width: 250;  
 -fx-min-height:70;  
}  
#closeAdd{  
 -fx-min-width: 250;  
 -fx-min-height:70;  
}  
#closeExit{  
 -fx-min-width: 250;  
 -fx-min-height:70;  
}  
#queueButtons{  
 -fx-min-width: 240;  
 -fx-min-height: 50;  
 -fx-max-width: 240;  
 -fx-max-height: 50;  
}

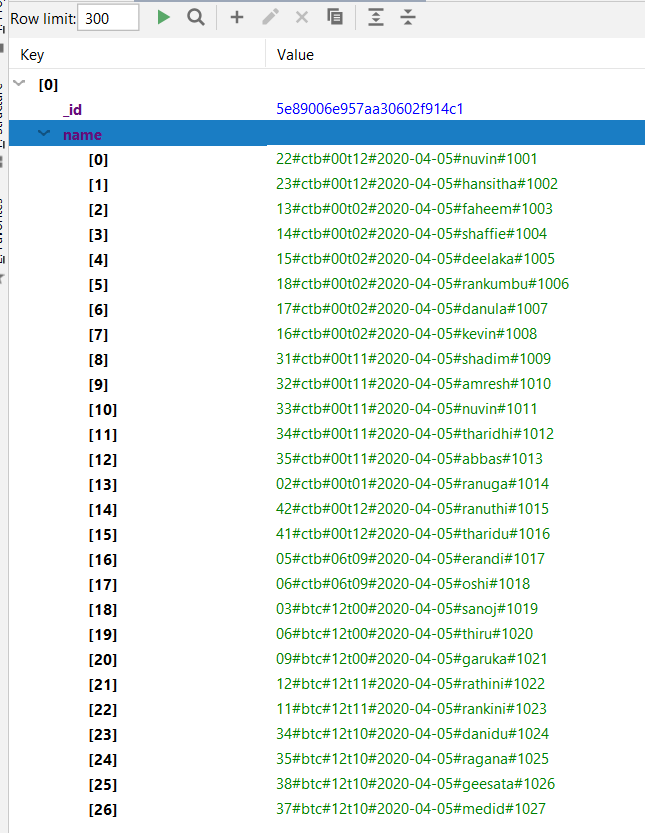
XML File for Database Connection

This program code implements and uses Mongo NoSQL Database for loading from the database to connect from coursework one.

Source-Code

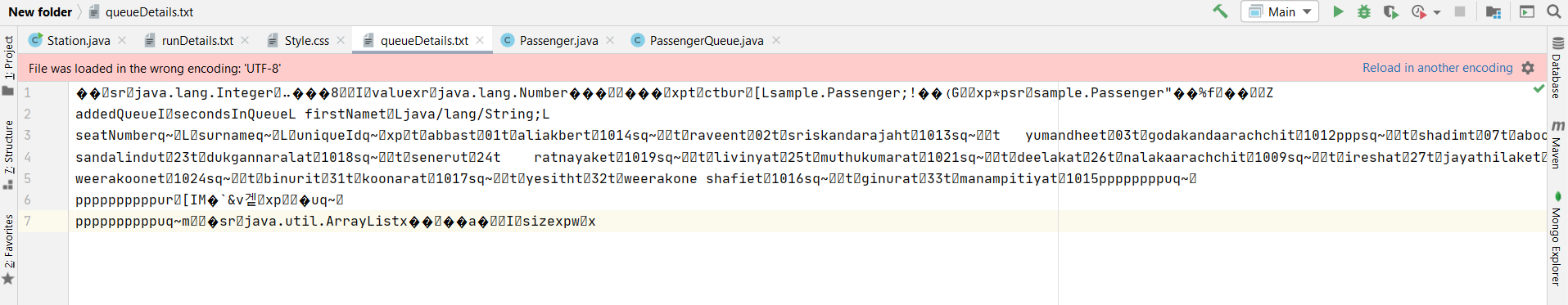
<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>groupId</groupId>  
 <artifactId>CourseWork</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.mongodb</groupId>  
 <artifactId>mongodb-driver</artifactId>  
 <version>3.12.2</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

Output- Database XML



**Img 11.1-** Hwo the outputs will appear on the mongo Explorer

Serialized file storing – Serialized data store



The complete final source code – Station.java

package sample;  
  
import com.mongodb.client.\*;  
import com.sun.scenario.effect.impl.sw.sse.SSEBlend\_SRC\_OUTPeer;  
import javafx.application.Application;  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.ScrollPane;  
import javafx.scene.layout.\*;  
import javafx.scene.text.Font;  
import javafx.scene.text.FontPosture;  
import javafx.scene.text.FontWeight;  
import javafx.stage.Stage;  
import javafx.stage.StageStyle;  
import org.bson.Document;  
import java.io.\*;  
import java.time.LocalDate;  
import java.util.\*;  
  
public class Station extends Application {  
  
 private static ArrayList<String> seatList = new ArrayList<>();  
 private static Passenger[] waitingRoom = new Passenger[42];  
 private static PassengerQueue trainQueueOne = new PassengerQueue();  
 private static PassengerQueue trainQueueTwo = new PassengerQueue();  
 private static ArrayList<Passenger> boardedToTrain = new ArrayList<>();  
 private static final String[] stops={"Colombo-Fort","Polgahawela","Peradeniya","Gampola","Nawalapitiya","Hatton",  
 "Thalawakele","Nanuoya","Haputale","Diyatalawa","Bandarawela","Ella","Badulla"};  
 private static int station= 0;  
 private static String direction= null;  
  
 public static void main(String[] args) { launch(args); }  
  
 public void start(Stage primaryStage) {  
 LocalDate date = LocalDate.now();  
 Scanner input = new Scanner(System.in);  
 System.out.println("==================Welcome to Denuwara Menike Train Terminal!==================");  
 System.out.println(" "); //pretty prints for the gui to look pleasant  
 boolean found=false;  
 do {  
 System.out.println("Lets start up the terminal for today, "+date);  
 System.out.println("Press <L> to load existing information, or <M> to move to the Check In Window");  
 String option = input.next().toLowerCase(); //getting an input from user to check if he wants to load or check in  
 if (option.equals("l")) {  
 found=true;  
 System.out.println("Loading Data from file. Please Hang on!");  
 load(); //calls UDF(user defined function) load if input is l  
 } else if (option.equals("m")) {  
 found=true;  
 checkIn(); //calls UDF check in to open up giu if input is m  
 } else { //if the input is un-identified  
 System.out.println("Oops! Incorrect input! Please reconsider input. ");  
 }  
 }while(!found);  
 }  
  
 //=================================================== CHECK IN ===================================================//  
 public static void checkIn() {  
  
 //============================================================================================================//  
 // INITIALIZING ELEMENTS  
  
 //------------------------------------------------------------------------------------------------------- Stages  
  
 ArrayList<String> newList = new ArrayList<>();  
  
 Stage stageOne = new Stage();  
 BorderPane rootOne = new BorderPane();  
 Scene sceneOne = new Scene(rootOne,700,250);  
 stageOne.setScene(sceneOne);  
 rootOne.getStylesheets().add("/style.css");  
 stageOne.setResizable(false);  
 stageOne.setTitle("Station Selection");  
 stageOne.show();  
  
 Stage stageTwo = new Stage();  
 BorderPane rootTwo = new BorderPane();  
 Scene sceneTwo = new Scene(rootTwo,1000,800);  
 stageTwo.setScene(sceneTwo);  
 rootTwo.getStylesheets().add("/style.css");  
 stageTwo.setResizable(false);  
 stageTwo.setTitle("Denuwara Menike Terminal- Self Check In");  
 stageTwo.initStyle(StageStyle.UNDECORATED); // Making the window undecorated so that the check in box cannot be  
 // cancelled unwillingly  
  
 LocalDate date = LocalDate.now(); //getting current date  
  
 //------------------------------------------------------------------------------------------------------- Labels  
 Label mainLabel = new Label("Denuwara Menike Terminal- Check-In Window");  
 mainLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 30));  
 mainLabel.setPadding(new Insets(40,20,20,20));  
  
 Label subLabel = new Label("Please select the 'Check In' Button co-responding with your seat");  
 subLabel.setFont(Font.font("sans-serif", FontPosture.REGULAR, 18));  
 subLabel.setPadding(new Insets(0,20,20,20));  
  
 Label stationMaster = new Label("Station Master, select the direction of the train and the Station Name");  
 stationMaster.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 18));  
 stationMaster.setPadding(new Insets(40,20,5,20));  
  
  
 Label stationMasterFooter = new Label("Today's date: "+date);  
  
 Label deets = new Label("Denuwara Menike Train Terminal; Station Name: "+stops[station]+  
 "; Date: "+date);  
  
  
 //------------------------------------------------------------------------------------------------------ Buttons  
  
 Button stationSubmit = new Button("OK");  
 stationSubmit.setId("stationButtons");  
  
 Button stationClear = new Button("Clear Selections");  
 stationClear.setId("stationButtons");  
  
 Button closeCheckIn = new Button("Close Check In");  
 closeCheckIn.setId("closeCheckIn");  
  
 //-------------------------------------------------------------------------------------------------- Combo Boxes  
  
 ComboBox selectDirection = new ComboBox();  
 selectDirection.setId("combo");  
 selectDirection.getItems().addAll("Colombo To Badulla", "Badulla to Colombo");  
 selectDirection.setPromptText("Select Direction");  
  
 ComboBox selectStation = new ComboBox();  
 selectStation.setId("combo");  
 selectStation.setPromptText("Select Station");  
  
  
  
 //-------------------------------------------------------------------------------------------------- Alert Boxes  
  
 Alert confirmClose = new Alert(Alert.AlertType.CONFIRMATION);  
 confirmClose.setHeaderText("Confirm Close");  
 confirmClose.setTitle("Are you sure to close the Check In counter?");  
 confirmClose.setContentText("Closing this will not allow anymore people to check in");  
  
 Alert confirmCheckIn = new Alert(Alert.AlertType.CONFIRMATION);  
 confirmCheckIn.setTitle("Confirm Check In");  
 confirmCheckIn.setContentText("Make sure you've selected the correct Check-In box");  
  
 //============================================================================================================//  
 // GUI INITIALIZATION  
  
 //--------------------------------------------------------------------------------------Stage One GUI Components  
  
 //-----Main boxes used  
 VBox headerOne = new VBox(20);  
 HBox headerLineOne = new HBox(20);  
 HBox footerLineOne = new HBox(20);  
  
 //-----Alignment of the main boxes  
 headerOne.setAlignment(Pos.CENTER);  
 headerLineOne.setAlignment(Pos.CENTER);  
 footerLineOne.setAlignment(Pos.CENTER);  
  
 rootOne.setTop(headerOne);  
  
 //-----putting in labels into the header  
 headerOne.getChildren().add(stationMaster);  
 headerOne.getChildren().add(headerLineOne);  
 headerOne.getChildren().add(footerLineOne);  
 headerOne.getChildren().add(stationMasterFooter);  
  
 headerLineOne.getChildren().add(selectDirection);  
 headerLineOne.getChildren().add(selectStation);  
 footerLineOne.getChildren().add(stationSubmit);  
 footerLineOne.getChildren().add(stationClear);  
  
 //-----default disabling elements  
 selectStation.setDisable(true);  
 stationSubmit.setDisable(true);  
 stationClear.setDisable(true);  
  
  
 //--------------------------------------------------------------------------------------Stage Two GUI Components  
  
 //-----Main boxes used  
 VBox headerTwo = new VBox(10);  
 HBox centerTwo = new HBox(75);  
 VBox footerTwo = new VBox(20);  
  
 // creating a scroll pane to display the names  
 ScrollPane centerScroll = new ScrollPane(centerTwo);  
  
 VBox names = new VBox(30);  
 VBox ids = new VBox(30);  
 VBox buttons = new VBox(20);  
  
 //-----Initializing elements  
 names.setPadding(new Insets(14,0,0,0));  
 ids.setPadding(new Insets(18,0,0,0));  
 buttons.setPadding(new Insets(14,0,0,0));  
 centerScroll.setPadding(new Insets(12,0,12,0));  
  
 rootTwo.setTop(headerTwo);  
 rootTwo.setCenter(centerScroll);  
 rootTwo.setBottom(footerTwo);  
  
 headerTwo.setAlignment(Pos.CENTER);  
 centerTwo.setAlignment(Pos.CENTER);  
 footerTwo.setAlignment(Pos.CENTER);  
  
 names.setAlignment(Pos.BASELINE\_RIGHT);  
 ids.setAlignment(Pos.CENTER);  
  
 centerScroll.setContent(centerTwo);  
 centerScroll.setFitToWidth(true);  
  
 headerTwo.getChildren().add(mainLabel);  
 headerTwo.getChildren().add(subLabel);  
  
 centerTwo.getChildren().add(names);  
 centerTwo.getChildren().add(ids);  
 centerTwo.getChildren().add(buttons);  
  
 for (int i=0;i<=12;i++){  
 selectStation.getItems().add(stops[i]); // populating comboBox using array- stops  
 }  
  
 selectDirection.valueProperty().addListener((observable, oldValue, newValue) -> {  
  
 selectStation.setDisable(false); //disabling already selected buttons  
 selectDirection.setDisable(true);  
 stationClear.setDisable(false);  
  
 selectStation.valueProperty().addListener(((observable1, oldValue1, newValue1) -> {  
 for (int i=0;i<=12;i++){  
 if (stops[i].equals(newValue1.toString())){ //traversing array to get selected value index  
 station=i; //storing index of value selected  
 }  
  
 if (newValue.toString().contains("Colombo To Badulla")){  
 direction="ctb"; //initializing global variables according to inputs  
 }else{  
 direction="btc";  
 }  
 }  
  
 stationSubmit.setDisable(false);  
  
 }));  
 });  
  
 stationSubmit.setOnAction(event -> {  
 stageOne.close();  
 stageTwo.show();  
  
 System.out.println("Getting your data from the System. Hold Tight!");  
 MongoClient client = MongoClients.create();  
 MongoDatabase dataBase = client.getDatabase("BookingDB");  
 MongoCollection<Document> baseCollection = dataBase.getCollection("bookCollection");  
 Document tempHold = baseCollection.find().first();  
 Object seats = tempHold.get("name");  
 seatList = (ArrayList<String>) seats;  
 System.out.println("Data retrieval Successful!");  
  
 for (int i = seatList.size() - 1; i >= 0; i--) { //checking for old data and deleting them  
 String current = seatList.get(i).substring(13, 23);  
 String curDirection = seatList.get(i).substring(3, 6);  
 int curStation = Integer.parseInt(seatList.get(i).substring(7,9));  
 String strDate = date.toString();  
 if (current.compareTo(strDate) != 0 || (!(curDirection.equals(direction))) || (curStation!=station)) {  
 seatList.remove(i);  
 }  
 }  
  
 Button[] checkIn = new Button[seatList.size()];  
  
 for (int i=0;i<seatList.size();i++){  
 // creating labels of data in order or visual ease  
 //retrieving data from concatted strings received from database  
 String fullName= seatList.get(i).substring(24,seatList.get(i).length()-5);  
 String id= seatList.get(i).substring(seatList.get(i).length()-4);  
 String[] name=fullName.split(" ", 2);  
 String firstName= name[0].substring(0,1).toUpperCase()+name[0].substring(1);  
 String secondName= name[1].substring(0,1).toUpperCase()+name[1].substring(1);  
  
 //creating buttons and labels, and putting values into them  
 names.getChildren().add(new Label(firstName+" "+secondName));  
 ids.getChildren().add(new Label(id));  
 checkIn[i]=new Button(" Click to Check In");  
 checkIn[i].setId("checkInButtons");  
 buttons.getChildren().add(checkIn[i]);  
 }  
  
 for (int i=0;i<seatList.size();i++){  
 int finalI = i;  
  
 checkIn[i].setOnAction(event1 -> { // set on action for each check in button  
  
 String wholeName = seatList.get(finalI).substring(24, seatList.get(finalI).length() - 5);  
 String uniqueID = seatList.get(finalI).substring(seatList.get(finalI).length() - 4);  
 String seatNumber = seatList.get(finalI).substring(0,2);  
 String[] splitNames = wholeName.split(" ", 2);  
  
 // changing the header text area to name of passenger for easy verification  
 confirmCheckIn.setHeaderText(wholeName);  
  
 Optional<ButtonType> result = confirmCheckIn.showAndWait(); // confirm check in request  
 if (result.get() == ButtonType.OK) {  
 checkIn[finalI].setDisable(true);  
 checkIn[finalI].setText("Checked In- Successful");  
  
 newList.add(seatList.get(finalI));  
 seatList.set(finalI, "null");  
 waitingRoom[Integer.parseInt(seatNumber)] = new Passenger(splitNames[0], splitNames[1], uniqueID, seatNumber);  
 //upon confirming, will be added to waiting list  
  
 }  
  
 });  
 }  
 });  
  
 //getting exit buttons and footer information to footer  
 footerTwo.getChildren().add(closeCheckIn);  
 footerTwo.getChildren().add(deets);  
  
 closeCheckIn.setOnAction(event -> {  
 Optional<ButtonType> result = confirmClose.showAndWait(); // confirming exit request  
 if (result.get() == ButtonType.OK) {  
 stageTwo.close(); //upon confirming, stage will be closed  
 try {  
 int count = 0;  
 for (int i=seatList.size()-1;i>=0;i--){  
 if (seatList.get(i).equals("null")){ //emptying seatlist  
 seatList.remove(i);  
 }else{  
 count++;  
 }  
 }  
 if (count>0) { // looking for number of passengers that didnt check in  
 System.out.println(count+" passenger(s) have not checked in.");  
 }else{  
 System.out.println("All passengers checked in!");  
 }  
 menu(); //calling back menu to sustain program- On press of exit button  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 stationClear.setOnAction(event -> {  
 stageOne.close(); //if clear selections button is pressed  
 try {  
 checkIn();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 });  
  
 }  
  
 //================================================== MAIN MENU ===================================================//  
 public static void menu() {  
 Scanner input = new Scanner(System.in);  
 String option;  
 System.out.println(" ");  
 System.out.println("==================================MAIN MENU===================================");  
 System.out.println(" ");  
 System.out.println(" ‘A’ : Add a passenger to the trainQueue");  
 System.out.println(" ‘V’ : View the trainQueue");  
 System.out.println(" ‘D’ : Delete passenger from the trainQueue");  
 System.out.println(" ‘S’ : Store trainQueue data");  
 System.out.println(" ‘L’ : Load data back");  
 System.out.println(" ‘R’ : Run the simulation and produce report");  
 System.out.println(" 'Q' : Exit program");  
 System.out.print("Enter your option here: ");  
 option = input.next(); // receives input from user  
 option = option.toUpperCase();  
 switch (option) {  
 case "A":  
 addPassenger(); //call add customers and evokes GUI  
 break;  
 case "V":  
 view(); //call view all seats and evokes GUI  
 break;  
 case "D":  
 System.out.println("");  
 System.out.println("======================= Delete Passenger =======================");  
 System.out.println("");  
 delete(); //call delete customer data  
 break;  
 case "S":  
 System.out.println("");  
 System.out.println("======================= Store from File ========================");  
 System.out.println("");  
 store(); //call store data  
 break;  
 case "L":  
 System.out.println("");  
 System.out.println("======================== Load from Fle =========================");  
 System.out.println("");  
 load(); //call load data  
 break;  
 case "R":  
 System.out.println("");  
 System.out.println("======================== Run Simulation ========================");  
 System.out.println("");  
 run(); //call Simulaton  
 break;  
 case "Q":  
 System.out.println("Oh No! Are you sure to exit? Press <B> to cancel exit, or any other key to exit: ");  
 String choice= input.next().toLowerCase(); // getting users confirmation for exit  
 if (choice.equals("b")){ // if user chose to stay in program  
 menu();  
 }else { //if user chose to exit program  
 System.out.println("Thank you using Denuwara Manike Train Terminal System. " +  
 "Have a good day and a Safe Journey!");  
 System.exit(1);  
 }  
 default:  
 System.out.println("Oops! We couldn't read that. Please check the options again and re-enter.");  
 menu();  
 }  
 }  
  
 //=============================================== ADD PASSENGERS =================================================//  
 public static void addPassenger(){  
  
 //============================================================================================================//  
 // INITIALIZING ELEMENTS  
  
 //------------------------------------------------------------------------------------------------------- Stages  
  
 Stage stage = new Stage();  
 BorderPane root = new BorderPane();  
 Scene scene = new Scene(root,1700,900);  
 stage.setScene(scene);  
 root.getStylesheets().add("/style.css");  
 stage.setResizable(false);  
 stage.setTitle("Denuwara Menike Terminal- ");  
  
 LocalDate date = LocalDate.now(); //local date  
  
 //---------------------------------------------------------------------------------------------------Alert Boxes  
  
 Alert noValues = new Alert(Alert.AlertType.WARNING);  
 noValues.setTitle("End of values");  
 noValues.setHeaderText("No more values to Add");  
 noValues.setContentText("No more values available to add to train queue");  
  
 Alert fullError = new Alert(Alert.AlertType.WARNING);  
 fullError.setTitle("Queue Full");  
 fullError.setHeaderText("Queue is filled up");  
 fullError.setContentText("The queue is already full");  
  
 //--------------------------------------------------------------------------------------------------------Labels  
  
 Label mainLabel = new Label("Denuwara Menike Terminal - Train Queue and Waiting Room");  
 mainLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 30));  
 mainLabel.setPadding(new Insets(60,20,5,20));  
  
 Label subLabel = new Label("Moving to function 'R' will move passengers to the train");  
 subLabel.setFont(Font.font("sans-serif", FontPosture.REGULAR, 18));  
 subLabel.setPadding(new Insets(5,20,5,20));  
  
 Label queueLabel = new Label("Train Queue Waiting to Board");  
 queueLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 22));  
 queueLabel.setPadding(new Insets(5,20,30,200));  
 queueLabel.setAlignment(Pos.CENTER);  
  
 Label waitingLabel = new Label("Waiting Room");  
 waitingLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 22));  
 waitingLabel.setPadding(new Insets(5,20,5,120));  
  
 Label queueOneLabel = new Label("Train Queue One");  
 queueOneLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,18));  
 queueOneLabel.setPadding(new Insets(0,0,5,0));  
  
 Label queueTwoLabel = new Label("Train Queue Two");  
 queueTwoLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,18));  
 queueTwoLabel.setPadding(new Insets(0,0,5,0));  
  
 Label deets = new Label("Denuwara Menike Train Terminal; Station Name: "+stops[station]+  
 "; Date: "+date);  
  
 //-------------------------------------------------------------------------------------------------------Buttons  
  
 Button exit = new Button(" Exit ");  
 exit.setId("closeExit");  
  
 //============================================================================================================//  
 // GUI INITIALIZATION  
  
 //------------------------------------------------------------------------------------------Stage GUI Components  
  
 //main boxes used  
 VBox header = new VBox();  
 VBox left = new VBox(5);  
 VBox center = new VBox();  
 VBox right = new VBox(5);  
 VBox footer = new VBox(10);  
  
 //-----------extra panes needed for both waiting room and queues  
  
 // used  
 VBox queueBoxOne = new VBox(5);  
 VBox queueBoxTwo = new VBox(5);  
  
 //used for displaying waiting room seats  
 FlowPane waitingSeatsOne = new FlowPane(15,15);  
 FlowPane waitingSeatsTwo = new FlowPane(15,15);  
  
 // setting boxes intitialized into box plot settings  
 root.setTop(header);  
 root.setLeft(left);  
 root.setCenter(center);  
 root.setRight(right);  
 root.setBottom(footer);  
  
 header.setAlignment(Pos.CENTER);  
 left.setAlignment(Pos.CENTER);  
 center.setAlignment(Pos.CENTER);  
 right.setAlignment(Pos.CENTER);  
 footer.setAlignment(Pos.CENTER);  
  
 FlowPane innerCenter = new FlowPane(); //used for waiting room gui  
 HBox innerLeft = new HBox();  
  
 innerLeft.setAlignment(Pos.CENTER);  
 innerCenter.setAlignment(Pos.CENTER);  
 queueBoxOne.setAlignment(Pos.CENTER);  
 queueBoxTwo.setAlignment(Pos.CENTER);  
  
 queueBoxOne.setPadding(new Insets(10, 0, 10, 200));  
 queueBoxTwo.setPadding(new Insets(10, 0, 10, 70));  
  
 //setting children  
 header.getChildren().add(mainLabel);  
 header.getChildren().add(subLabel);  
 left.getChildren().add(queueLabel);  
 left.getChildren().add(innerLeft);  
 center.getChildren().add(waitingLabel);  
 center.getChildren().add(innerCenter);  
 footer.getChildren().add(exit);  
  
 innerLeft.getChildren().add(queueBoxOne);  
 innerLeft.getChildren().add(queueBoxTwo);  
  
 innerCenter.getChildren().add(waitingSeatsOne);  
 innerCenter.getChildren().add(waitingSeatsTwo);  
  
 queueBoxOne.getChildren().add(queueOneLabel);  
 queueBoxTwo.getChildren().add(queueTwoLabel);  
  
 //--------------------------------------------------------------------------------- Program code controlling gui  
  
 Button[] queueButtons = new Button[20];  
  
 Random rand = new Random(); //importing random class  
 int num = rand.nextInt(6) + 1; // adding one to omit the occurance of o  
  
 for (int i=1;i<=num;i++){  
 boolean endLoop= false; //used to check when to exit the loop mentioned below  
  
 for (int count=0;count<42;count++){ //traverses whole array  
 int queueOne= trainQueueOne.getLength();  
 int queueTwo= trainQueueTwo.getLength();  
  
 //if condition checks if the specific value in the waiting room is not null  
 // also if it is not added9(according to the tag used to verify)  
 // train queue is not full  
 // also compares train queue sizes to enter values into the one with lesser passengers  
 if (waitingRoom[count]!=null && !waitingRoom[count].getAdded() && !trainQueueOne.isFull() &&  
 (queueTwo>queueOne || queueOne==queueTwo)){  
 trainQueueOne.add(waitingRoom[count]); //add the value from waiting room to train queue  
 waitingRoom[count].setAdded(true); //edits tag to true - ie. used  
 int diceOne= rand.nextInt(6)+1; // radomizing 3 more dice to get a random number to work as  
 int diceTwo= rand.nextInt(6)+1; // time taken  
 int diceThree= rand.nextInt(6)+1;  
 int time= diceOne+diceTwo+diceThree;  
 trainQueueOne.setTime(time); //using set time in the passenger queue class to set time  
 break;  
 }else if (waitingRoom[count]!=null && !waitingRoom[count].getAdded() && !trainQueueTwo.isFull() &&  
 queueOne>queueTwo){  
 trainQueueTwo.add(waitingRoom[count]); //add the value from waiting room to train queue  
 waitingRoom[count].setAdded(true); //edits tag to true - ie. used  
 int diceOne= rand.nextInt(6)+1; // radomizing 3 more dice to get a random number to work as  
 int diceTwo= rand.nextInt(6)+1; // time taken  
 int diceThree= rand.nextInt(6)+1;  
 int time= diceOne+diceTwo+diceThree;  
 trainQueueTwo.setTime(time); //using set time in the passenger queue class to set time  
 break;  
 }else if (trainQueueTwo.isFull() && trainQueueOne.isFull()){  
 System.out.println("All values added"); //if train queues are full , loop will be broken  
 break;  
 }else if(count==41){  
 endLoop=true;  
 noValues.showAndWait(); // if all values are added, messages will be prompted  
 break;  
 }  
 }  
 if (trainQueueTwo.isFull() && trainQueueOne.isFull()){  
 fullError.showAndWait(); // if both queues are full, again messages will be propmpted  
 break;  
 }  
 if (endLoop){  
 break; //if all values have been added loop will be broken again  
 }  
 }  
  
 // ---------------Train Queue gui program codes  
  
 if (trainQueueOne.isEmpty()){  
 Button emptyButton=new Button("Empty Train Queue"); // default buttons, sayaing empty  
 emptyButton.setId("queueButtons");  
 queueBoxOne.getChildren().add(emptyButton);  
 }else {  
 for (int i = 0; i < trainQueueOne.getLength(); i++) {  
 queueButtons[i] = new Button(trainQueueOne.accessName(i)); // names will be added to the button text  
 queueButtons[i].setId("queueButtons");  
 queueBoxOne.getChildren().add(queueButtons[i]);  
 }  
 }  
  
 if (trainQueueTwo.isEmpty()){  
 Button emptyButton=new Button("Empty Train Queue");// default buttons, sayaing empty  
 emptyButton.setId("queueButtons");  
 queueBoxTwo.getChildren().add(emptyButton);  
 }else {  
 for (int i = 0; i < trainQueueTwo.getLength(); i++) {  
 queueButtons[i] = new Button(trainQueueTwo.accessName(i));// names will be added to the button text  
 queueButtons[i].setId("queueButtons");  
 queueBoxTwo.getChildren().add(queueButtons[i]);  
 }  
 }  
  
 // --------------Waiting room giu program codes  
  
 Button[] waitingButtons = new Button[42];  
  
 //empty buttons firstly created  
 waitingSeatsOne.setPrefWrapLength(270);  
 waitingSeatsOne.setPadding(new Insets(70, 10, 10, 100));  
 for (int i=0;i<21;i++){  
 waitingButtons[i]=new Button("Empty"); // empty text  
 waitingButtons[i].setId("waitingButtons");  
 waitingSeatsOne.getChildren().add(waitingButtons[i]);  
 }  
  
 waitingSeatsTwo.setPrefWrapLength(270);  
 waitingSeatsTwo.setPadding(new Insets(70, 10, 10, 20));  
 for (int i=21;i<42;i++){  
 waitingButtons[i]=new Button("Empty"); //empty text  
 waitingButtons[i].setId("waitingButtons");  
 waitingSeatsTwo.getChildren().add(waitingButtons[i]);  
 }  
  
 // then will be colored  
 for (int i=0;i<42;i++){  
 if (waitingRoom[i]!=null && !waitingRoom[i].getAdded()){ //i+"\n Empty"  
 waitingButtons[Integer.parseInt(waitingRoom[i].getSeat())].setText(waitingRoom[i].getSeat()+"\n"+  
 waitingRoom[i].getName());  
 waitingButtons[Integer.parseInt(waitingRoom[i].getSeat())].setStyle("-fx-background-color: #c29ecd");  
 int finalI = i;  
 waitingButtons[i].setOnAction(event -> { // on click of these buttons will show a messsage box with all  
 // the passenger details  
 Alert info = new Alert(Alert.AlertType.INFORMATION);  
 info.setTitle("Passenger Details");  
 info.setHeaderText("Name: "+waitingRoom[finalI].getFullName());  
 info.setContentText("Seat Number: "+waitingRoom[finalI].getSeat()+"\n"+  
 "Unique ID: "+waitingRoom[finalI].getId());  
 info.showAndWait(); //showing the information box  
 });  
 }  
 }  
  
  
 stage.show();  
 footer.getChildren().add(deets);  
  
 exit.setOnAction(event -> { //closing the stage through exit button  
 stage.close();  
 menu();  
 });  
 }  
  
 //=============================================== VIEW PASSENGERS ================================================//  
 public static void view(){  
  
 //============================================================================================================//  
 // INITIALIZING ELEMENTS  
  
 //------------------------------------------------------------------------------------------------------- Stages  
  
 Stage stageOne = new Stage();  
 BorderPane rootOne = new BorderPane();  
 Scene sceneOne = new Scene(rootOne,800,800);  
 stageOne.setScene(sceneOne);  
 rootOne.getStylesheets().add("/style.css");  
 stageOne.setResizable(false);  
 stageOne.setTitle("Station Selection");  
 stageOne.show();  
  
 LocalDate date = LocalDate.now(); //local date  
  
 //------------------------------------------------------------------------------------------------------- Labels  
  
 Label mainLabel = new Label("Denuwara Menike Terminal - Waiting Room");  
 mainLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 30));  
 mainLabel.setPadding(new Insets(60,20,5,20));  
  
 Label subLabel = new Label("Select a seat to view more Information");  
 subLabel.setFont(Font.font("sans-serif", FontPosture.REGULAR, 18));  
 subLabel.setPadding(new Insets(5,20,5,20));  
  
 Label deets = new Label("Denuwara Menike Train Terminal; Station Name: "+stops[station]+  
 "; Date: "+date);  
  
  
 //------------------------------------------------------------------------------------------------------ Buttons  
  
 Button exit = new Button("Exit");  
 exit.setId("closeViewButton");  
  
  
 //============================================================================================================//  
 // GUI INITIALIZATION  
  
 //------------------------------------------------------------------------------------------Stage GUI Components  
  
 //main three boxes  
 VBox headerOne = new VBox();  
 HBox centerOne = new HBox(50);  
 VBox footerOne = new VBox(20);  
  
 //sub boxes  
 VBox centerLeft = new VBox(50);  
 VBox centerMid = new VBox(50);  
 VBox centerRight = new VBox();  
  
 //flowpanes for waiting room  
 FlowPane left = new FlowPane(15,15);  
 FlowPane mid = new FlowPane(15,15);  
  
 //positioning in border pane  
 rootOne.setTop(headerOne);  
 rootOne.setLeft(centerOne);  
 rootOne.setBottom(footerOne);  
  
 headerOne.setAlignment(Pos.CENTER);  
 centerOne.setAlignment(Pos.CENTER);  
 footerOne.setAlignment(Pos.CENTER);  
  
 //adding labels and buttons using getchildern  
 headerOne.getChildren().add(mainLabel);  
 headerOne.getChildren().add(subLabel);  
  
 centerOne.getChildren().add(centerLeft);  
 centerOne.getChildren().add(centerMid);  
 centerOne.getChildren().add(centerRight);  
  
 centerLeft.getChildren().add(left);  
 centerMid.getChildren().add(mid);  
  
 //------------ Waiting room gui program code  
 Button[] waitingButtons = new Button[42];  
  
 left.setPrefWrapLength(270); // setting a max width for box  
 left.setPadding(new Insets(70, 10, 10, 100));  
  
 for (int i=0;i<21;i++){  
 waitingButtons[i]=new Button("Empty"); //default buttons with text empty  
 waitingButtons[i].setId("waitingButtons");  
 waitingButtons[i].setStyle("-fx-background-color: #4f65a8");  
 left.getChildren().add(waitingButtons[i]); //added to root  
 }  
  
 mid.setPrefWrapLength(270);  
 mid.setPadding(new Insets(70, 10, 10, 20));  
 for (int i=21;i<42;i++){  
 waitingButtons[i]=new Button("Empty"); //default buttons with text empty  
 waitingButtons[i].setId("waitingButtons");  
 waitingButtons[i].setStyle("-fx-background-color: #4f65a8");  
 mid.getChildren().add(waitingButtons[i]); //added to root  
 }  
  
 for (int i=0;i<42;i++){  
 if (waitingRoom[i]!=null){ // if the waiting index has values  
 waitingButtons[Integer.parseInt(waitingRoom[i].getSeat())].setText(waitingRoom[i].getSeat()+"\n"+  
 waitingRoom[i].getName());  
 waitingButtons[Integer.parseInt(waitingRoom[i].getSeat())].setStyle("-fx-background-color: #c29ecd");  
 int finalI = i;  
 waitingButtons[i].setOnAction(event -> { // when a button is clicked  
 Alert info = new Alert(Alert.AlertType.INFORMATION);  
 info.setTitle("Passenger Details");  
 info.setHeaderText("Name: "+waitingRoom[finalI].getFullName());  
 info.setContentText("Seat Number: "+waitingRoom[finalI].getSeat()+"\n"+  
 "Unique ID: "+waitingRoom[finalI].getId());  
 info.showAndWait();  
 });  
 }  
 }  
  
 footerOne.getChildren().add(exit);  
 footerOne.getChildren().add(deets);  
  
 exit.setOnAction(event -> {  
 stageOne.close();  
 menu();  
 });  
  
 }  
  
 //=============================================== DELETE PASSENGER ===============================================//  
 public static void delete(){  
  
 Scanner input = new Scanner(System.in);  
 System.out.println("Delete Options: ");  
 System.out.println(" 1. Delete from Train Queue and add back to Waiting room");  
 System.out.println(" 2. Delete from Train Queue and leave train Station");  
 System.out.println("Enter your option number to begin deletion: ");  
 String deleteType = input.next(); // checking if passenger wants to leave the train station or just the queue  
 switch (deleteType){  
 case "1": //if the selection is one  
 System.out.println("Enter your first name or Unique ID to delete: ");  
 String name= input.next().toLowerCase();// name will be made to lowercase for verification  
 boolean found= trainQueueOne.delete(name); // checks for name in delete fucntion in passenger queue class  
 boolean nextFound;  
 if (!found){ // if boolean returned from queue one search is false  
 nextFound= trainQueueTwo.delete(name); //check in other queue  
 if (!nextFound){// if still failed t find, propmts no name found  
 System.out.println("Oops! Couldn't find that! Want to give it another try? ");  
 System.out.println("Press <O> to try again, or any other key to move back to the main menu: ");  
 String choice= input.next().toLowerCase();  
 if (choice.equals("o")){ // gives a chance for the user to start over  
 delete();  
 }else{  
 System.out.println("Process Exit. No changes to local.");  
 menu(); //calling back menu to sustain program  
 }  
 }else{  
 System.out.println("Process Exit. Local updated. Make sure to press upload the data.");  
 menu(); //calling back menu to sustain program- On press of exit button  
 }  
 }  
 break;  
 case "2":  
 System.out.println("Enter your first name or Unique ID to delete: ");  
 name= input.next().toLowerCase();// name will be made to lowercase for verification  
 found= trainQueueOne.delete(name); // checks for name in delete fucntion in passenger queue class  
 if (!found){ // if boolean returned from queue one search is false  
 nextFound= trainQueueTwo.delete(name); //check in other queue  
 if (!nextFound){// if still failed t find, propmts no name found  
 System.out.println("Oops! Couldn't find that! Want to give it another try? ");  
 System.out.println("Press <O> to try again, or any other key to move back to the main menu: ");  
 String choice= input.next().toLowerCase();  
 if (choice.equals("o")){ // gives a chance for the user to start over  
 delete();  
 }else{  
 System.out.println("Process Exit. No changes to local.");  
 menu(); //calling back menu to sustain program  
 break;  
 }  
 }else{  
 System.out.println("Process Exit. Local updated. Make sure to press upload the data.");  
 menu(); //calling back menu to sustain program- On press of exit button  
 break;  
 }  
 }  
 for (int i=0;i<42;i++){ // removes passenger from waiting room  
 if (waitingRoom[i] != null && (waitingRoom[i].getName().contains(name) ||  
 (waitingRoom[i].getId().equals(name)))) {  
 waitingRoom[i] = null; // removing as in making it null  
 break;  
 }  
 }  
 menu();  
 break;  
 default: // default will call back the function  
 System.out.println("Oops! That's an incorrect input!");  
 delete();  
 }  
 }  
  
 //===================================================== STORE ====================================================//  
 public static void store(){  
 try{  
 System.out.println("Store file Opened");  
 File storeQueue = new File("queueDetails.txt"); //opens file  
 FileOutputStream fileOutputStream = new FileOutputStream(storeQueue); //opens file for write  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream); //passes for objects  
 System.out.println("Store file Opened");  
  
 // values passed in  
 objectOutputStream.writeObject(station);  
 objectOutputStream.writeObject(direction);  
 objectOutputStream.writeObject(waitingRoom);  
 objectOutputStream.writeObject(trainQueueOne.getQueue());  
 objectOutputStream.writeObject(trainQueueOne.storeAdditionalAll());  
 objectOutputStream.writeObject(trainQueueTwo.getQueue());  
 objectOutputStream.writeObject(trainQueueTwo.storeAdditionalAll());  
 objectOutputStream.writeObject(boardedToTrain);  
  
 objectOutputStream.flush();  
 System.out.println("Store Successful");  
 objectOutputStream.close();  
 System.out.println("Store Closed: Success");  
 fileOutputStream.close();  
 } catch (IOException e) {  
 System.out.println("Error loading data to file!");  
 }  
 menu();  
 }  
  
 //===================================================== LOAD =====================================================//  
 public static void load(){  
 try{  
 System.out.println("Load file Opened");  
 File storeQueue = new File("queueDetails.txt");  
 FileInputStream fileInputStream = new FileInputStream(storeQueue);  
 ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);  
  
 station= (int) objectInputStream.readObject();  
 direction= (String) objectInputStream.readObject();  
 waitingRoom=((Passenger[]) objectInputStream.readObject());  
 trainQueueOne.setQueue((Passenger[]) objectInputStream.readObject());  
 trainQueueOne.initialize((int[]) objectInputStream.readObject());  
 trainQueueTwo.setQueue((Passenger[]) objectInputStream.readObject());  
 trainQueueTwo.initialize((int[]) objectInputStream.readObject());  
 boardedToTrain= (ArrayList<Passenger>) objectInputStream.readObject();  
 System.out.println("Load Complete");  
  
 System.out.println("Load Closed: Success");  
  
 }catch (IOException | ClassNotFoundException e) {  
 System.out.println("Error retrieving data from file");  
 }  
 menu();  
 }  
  
 //===================================================== RUN ======================================================//  
 public static void run(){  
  
 // adding to train from train queue - ie. boarding  
 boolean repeat=false;  
 if (trainQueueOne.getLength()!=0) {  
 do {  
 Passenger passenger = trainQueueOne.remove();  
 boardedToTrain.add(passenger);  
 System.out.println(passenger.getFullName() + " added to Train");  
 if (trainQueueOne.getLength() == 0) {  
 repeat = true;  
 }  
 } while (!repeat);  
 }  
  
 // adding to train from train queue - ie. boarding  
 repeat=false;  
 if (trainQueueTwo.getLength()!=0) {  
 do {  
 Passenger passenger = trainQueueTwo.remove();  
 boardedToTrain.add(passenger);  
 System.out.println(passenger.getFullName() + " added to Train");  
 if (trainQueueTwo.getLength() == 0) {  
 repeat = true;  
 }  
 } while (!repeat);  
 }  
  
 //============================================================================================================//  
 // INITIALIZING ELEMENTS  
  
 //------------------------------------------------------------------------------------------------------- Stages  
  
 Stage stage = new Stage();  
 BorderPane root = new BorderPane();  
 Scene scene = new Scene(root,1000,900);  
 stage.setScene(scene);  
 root.getStylesheets().add("/style.css");  
 stage.setResizable(false);  
 stage.setTitle("Denuwara Menike Terminal- Terminal Operations Summary Report");  
 stage.show();  
  
 LocalDate date = LocalDate.now(); //local date  
  
  
 //------------------------------------------------------------------------------------------------------- Labels  
  
 Label mainLabel = new Label("Denuwara Menike Terminal Operations Summary Report");  
 mainLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR, 30));  
 mainLabel.setPadding(new Insets(40,20,20,20));  
  
 Label deets = new Label("Denuwara Menike Train Terminal; Station Name: "+stops[station]+  
 "; Date: "+date);  
  
 Label dateInfo = new Label("Train boarding summary as at Date: ");  
 dateInfo.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
 dateInfo.setPadding(new Insets(0,20,0,20));  
  
 Label nowDate = new Label(date.toString());  
  
 Label directionLabel = new Label("Train Direction: ");  
 directionLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
 directionLabel.setPadding(new Insets(0,20,0,20));  
  
 Label side = new Label();  
 if (direction.equals("ctb")) {  
 side.setText("Colombo-Fort to Badulla ");  
 }else{  
 side.setText("Badulla to Colombo-Fort ");  
 }  
  
 Label stationLabel = new Label("Station Name: ");  
 stationLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
 stationLabel.setPadding(new Insets(0,20,0,20));  
  
 Label stationInfo = new Label(stops[station]);  
  
 Label queueOneLabel = new Label("Train Queue One");  
 queueOneLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,18));  
 queueOneLabel.setPadding(new Insets(0,0,5,0));  
  
 Label queueTwoLabel = new Label("Train Queue Two");  
 queueTwoLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,18));  
 queueTwoLabel.setPadding(new Insets(0,0,5,0));  
  
 //adding data from train queues to labels  
  
 Label queueOneShortest = new Label("Shortest Stay in Queue: "+trainQueueOne.getShortestStay());  
  
 Label queueOneLongest = new Label("Longest Stay in Queue: "+trainQueueOne.getLongestStay());  
  
 Label queueOneLength = new Label("Maximum length Attained: "+trainQueueOne.getLongestLength());  
  
 Label queueOneMaxStay = new Label("Total Time in Queue: "+trainQueueOne.getLongestStay());  
  
 Label queueOneAverage = new Label("Average Waiting Time in Queue: "+trainQueueOne.getAverage());  
  
  
 Label queueTwoShortest = new Label("Shortest Stay in Queue: "+trainQueueTwo.getShortestStay());  
  
 Label queueTwoLongest = new Label("Longest Stay in Queue: "+trainQueueTwo.getLongestStay());  
  
 Label queueTwoLength = new Label("Maximum length Attained: "+trainQueueTwo.getLongestLength());  
  
 Label queueTwoMaxStay = new Label("Total Time in Queue: "+trainQueueTwo.getLongestStay());  
  
 Label queueTwoAverage = new Label("Average Waiting Time in Queue: "+trainQueueTwo.getAverage());  
  
 Label fullName = new Label ("Full Name");  
 fullName.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
  
 Label id = new Label("Ticket ID");  
 id.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
  
 Label seatNumber = new Label("Seat Number");  
 seatNumber.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
  
 Label secondsLabel = new Label("Seconds In Queue");  
 secondsLabel.setFont(Font.font("sans-serif", FontWeight.BOLD, FontPosture.REGULAR,17));  
  
  
 //------------------------------------------------------------------------------------------------------ Buttons  
  
 Button exit = new Button("Exit");  
 exit.setId("closeRun");  
  
 //============================================================================================================//  
 // GUI PROGRAM CODES  
  
 //------------------------------------------------------------------------------------------------------- Stages  
  
 VBox header = new VBox(10); //main boxes  
 HBox center = new HBox(50);  
 ScrollPane centerScroll = new ScrollPane(center);  
  
 HBox headerSummary = new HBox(50); //used to show summary of queues  
  
 HBox dateBox = new HBox(); //used to show important aspects of the data  
 HBox directionBox = new HBox();  
 HBox stationBox = new HBox();  
  
 VBox queueOne = new VBox(2); //data of each queue  
 queueOne.setPadding(new Insets(5,0,20,0));  
 VBox queueTwo = new VBox(2);  
 queueTwo.setPadding(new Insets(5,0,20,0));  
  
 VBox names = new VBox(30);  
 VBox ids = new VBox(30);  
 VBox seat = new VBox(30);  
 VBox seconds = new VBox(30);  
  
 VBox footer = new VBox(20);  
 footer.setPadding(new Insets(20,0,0,0));  
  
 names.setPadding(new Insets(9,0,0,0));  
 ids.setPadding(new Insets(14,0,0,0));  
 seat.setPadding(new Insets(14,0,0,0));  
 seconds.setPadding(new Insets(11,0,0,0));  
  
 root.setTop(header);  
 root.setCenter(centerScroll);  
 root.setBottom(footer);  
  
 // adding children to get  
 header.setAlignment(Pos.CENTER);  
 header.getChildren().add(mainLabel);  
 header.getChildren().add(dateBox);  
 header.getChildren().add(directionBox);  
 header.getChildren().add(stationBox);  
 header.getChildren().add(headerSummary);  
  
 dateBox.setAlignment(Pos.CENTER);  
 dateBox.getChildren().add(dateInfo);  
 dateBox.getChildren().add(nowDate);  
  
 directionBox.setAlignment(Pos.CENTER);  
 directionBox.getChildren().add(directionLabel);  
 directionBox.getChildren().add(side);  
  
 stationBox.setAlignment(Pos.CENTER);  
 stationBox.getChildren().add(stationLabel);  
 stationBox.getChildren().add(stationInfo);  
  
 headerSummary.setAlignment(Pos.CENTER);  
 headerSummary.getChildren().add(queueOne);  
 headerSummary.getChildren().add(queueTwo);  
  
 // ----------------------adding the important data  
 // adding information of train queue one  
 queueOne.setAlignment(Pos.CENTER);  
 queueOne.getChildren().add(queueOneLabel);  
 queueOne.getChildren().add(queueOneShortest);  
 queueOne.getChildren().add(queueOneLongest);  
 queueOne.getChildren().add(queueOneLength);  
 queueOne.getChildren().add(queueOneMaxStay);  
 queueOne.getChildren().add(queueOneAverage);  
  
 //adding information of train queue two  
 queueTwo.setAlignment(Pos.CENTER);  
 queueTwo.getChildren().add(queueTwoLabel);  
 queueTwo.getChildren().add(queueTwoShortest);  
 queueTwo.getChildren().add(queueTwoLongest);  
 queueTwo.getChildren().add(queueTwoLength);  
 queueTwo.getChildren().add(queueTwoMaxStay);  
 queueTwo.getChildren().add(queueTwoAverage);  
  
 //adding labels for name, id. seat, seconds  
 center.setAlignment(Pos.CENTER);  
 center.getChildren().add(names);  
 center.getChildren().add(ids);  
 center.getChildren().add(seat);  
 center.getChildren().add(seconds);  
 centerScroll.setContent(center);  
  
 names.setAlignment(Pos.CENTER);  
 ids.setAlignment(Pos.CENTER);  
 seat.setAlignment(Pos.CENTER);  
 seconds.setAlignment(Pos.CENTER);  
 centerScroll.setFitToWidth(true);  
  
 names.getChildren().add(fullName);  
 ids.getChildren().add(id);  
 seat.getChildren().add(seatNumber);  
 seconds.getChildren().add(secondsLabel);  
  
 // this for loop will add values to the boxes  
 for (Passenger passenger : waitingRoom) {  
 if (passenger != null && passenger.getAdded()) {  
 names.getChildren().add(new Label (passenger.getFullName()));// adds to name box  
 ids.getChildren().add(new Label (passenger.getId())); // adds to id box  
 seat.getChildren().add(new Label(passenger.getSeat())); // adds to seat box  
 seconds.getChildren().add(new Label(String.valueOf(passenger.getSeconds()))); // adds to seconds box  
 }  
 }  
  
 footer.setAlignment(Pos.CENTER);  
 footer.getChildren().add(exit);  
 exit.setOnAction(event -> { //exit button soa  
 stage.close();  
 menu();  
 });  
 footer.getChildren().add(deets);  
  
 System.out.println(seatList.size());  
  
 try {  
 int longest=0;  
 for (int i=0;i<42;i++){  
 //Comparing sizes of the names to get longest  
 if (waitingRoom[i]!=null && waitingRoom[i].getFullName().length()>longest) {  
 longest=waitingRoom[i].getFullName().length();  
 }  
 }  
  
 //========================================================================================================//  
 // STORE TO FILE  
  
 // Format has been made according to eye patterns.  
 File storeQueue = new File("Report\_on\_"+date+".txt");  
 FileWriter writer = new FileWriter(storeQueue);  
 writer.write("Denuwara Menike Train Terminal");  
 writer.write("\r\n");  
 writer.write("Train boarding summary as at " + date);  
 writer.write("\r\n");  
 if (direction.equals("ctb")) {  
 writer.write("Train Direction: Colombo-Fort to Badulla ");  
 }else{  
 writer.write("Train Direction: Badulla to Colombo-Fort ");  
 }  
 writer.write("\r\n");  
 writer.write("Station Name: "+stops[station]);  
 writer.write("\r\n");writer.write("\r\n");  
 writer.write("Passenger Details: ");  
 writer.write("\r\n");  
 writer.write(" | Name");  
 for (int i=0;i<=longest-4;i++){ writer.write(" "); }  
 writer.write(" | Ticket ID | Seat Number | Time In Queue |");  
 writer.write("\r\n"); // Move to next line  
 writer.write(" |-----");  
 for (int i=0;i<=longest-4;i++){ writer.write("-"); }  
 writer.write("-|------------|--------------|---------------| ");  
 writer.write("\r\n");  
 for (Passenger passenger : waitingRoom) {  
 if (passenger != null && passenger.getAdded()) {  
 writer.write(" | ");  
 writer.write(passenger.getFullName());  
 for (int nextI = 0; nextI <= longest - passenger.getFullName().length(); nextI++) {  
 writer.write(" ");  
 }  
 writer.write(" | " + passenger.getId() + " ");  
 writer.write(" | " + passenger.getSeat() + " ");  
 if (passenger.getSeconds()<10){  
 writer.write(" | 0" + passenger.getSeconds() + " |");  
 }else{  
 writer.write(" | " + passenger.getSeconds() + " |");  
 }  
 writer.write("\r\n");  
 }  
 }  
 writer.write("\r\n");writer.write("\r\n");  
 writer.write("Processing Summary: ");  
 writer.write("\r\n");  
 writer.write(" | Shortest Stay in queue | Longest Stay in queue |" +  
 " Maximum length Attained | Total Time in Queue | Average Waiting Time in Queue |");  
 writer.write("\r\n");  
 writer.write("Train Queue 1 | "+trainQueueOne.getShortestStay()+ " | "+  
 trainQueueOne.getLongestStay()+ " | "+trainQueueOne.getLength()+  
 " | "+trainQueueOne.getMaxStay()+ " | "+  
 trainQueueOne.getAverage()+" |");  
 writer.write("\r\n");  
  
 writer.write("Train Queue 2 | "+trainQueueTwo.getShortestStay()+ " | "+  
 trainQueueTwo.getLongestStay()+ " | "+trainQueueTwo.getLength()+  
 " | "+trainQueueTwo.getMaxStay()+ " | "+  
 trainQueueTwo.getAverage()+" |");  
 writer.close();  
 }catch (IOException e) {  
 System.out.println("Error loading data to file.");  
 }  
 }  
  
}

# **Test Plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test Input** | **Expected Output** | **Actual Output** | **Pass/Fail** |
| 1. | Values other than A, V, D, S, L, R and Q | Error Message | Error message shown and will re-prompt for input without crashing | Pass |
| 2. | Input “a”/ “A” | Display GUI and add random number of passengers to queue | Function add customers (pages 18-26) and a random will be generated. This number of customers will be added to the train queue. Process is automatic (Further discussed below) | Pass |
| 3. | Input “v”/ “V” | Display GUI and show 42 slots with passenger name | Function view seats (pages 30-32) will be called. 42b seats will be shown, according to a color pattern. (Further discussed below) | Pass |
| 4. | Input “d”/ “D” | Prompt the user for name to delete and removes element from train queue | Function Delete Customer (pages 35-39) will be called. Will prompt the user asking do select delete option. Then, will prompt the user for name or unique ID. If name found, prints all details of passenger and prompts user to confirm deletion. Upon confirmation, will delete from local copy of the data. | Pass |
| 5. | Input “S”/ “s” | Save the current information of the passenger class to a text file. | Function Store (page 40) will be called. The text will be put into objects and stored as serialized data in a text file | Pass |
| 6. | Input “L”/ “l” | Load the data from text file to the current running data structures | Function Load (pages 41) will be called. Current data will be stored into text file. | Pass |
| 7. | Input “r”/ “R” | Use bubble sort to print array list in order | Function Run will open a giu report to the screen, showing all the details of the current train queues. The data will also be put into a file created according to the date. | Pass |
| 8. | Input “q”/ “Q” | Exit the program | System will remind user to store data, and will continue to end program. | Pass |
| Add to Queue Method testing | | | | |
| 9. | A pressed when no passengers are checked in/ waiting room is empty | Message should be shown that no data is available | A message box will be popped before the giu opens | Pass |
| 10. | A is pressed when the train queue is full | Message should be shown that queue is full | A message box will be popped before the giu opens | Pass |
| 11. | Empty seats selected | No data shown | No data shown | Pass |
| View Seats Method testing | | | | |
| 12. | Selection of Seats | Should not be allowed | All the seats will be disabled | Pass |
| 13. | Identification of seats | Should be distinguished | Seats will be colored according to a color code notified on the top of the stage | Pass |
| 14. | Selection of exit button | Should move back to menu | Will prompt user to confirm selection. Upon selection, will move to menu | Pass |
| Delete Customers Method testing | | | | |
| 15. | Characters entered instead of String | Should prompt error | Program also allows the user to enter an Integer, to allow them to deleted using their Unique ID. However, if number/ string is not available, error will be prompted saying “No Data Found” | Pass |
| 16. | Unavailable name entered | Should prompt error | Program will prompt and error to the user saying “No data Found” and prompt if they wish to re-try entering a name. | Pass |
| 17. | Upon entering normal data | Should remove data from array list and prompt user that it is deleted | Program will traverse through the array list and if the name/ unique ID is found, user will be prompted to confirm. Upon confirmation, data will be removed. Then moves to the main menu. | Pass |
| 18. | Upon selecting not to delete found data | Should cancel deletion | Will cancel the deletion and move back to the main menu. | Pass |
| Store Method Testing | | | | |
| All data in array list will be stored in database. | | | | |
| Load Method Testing | | | | |
| All data in database will be retrieved from database and stored in the Database. All data belonging to older dates will be automatically removed | | | | |

# **Self-Assessment**

**Criteria – Add Customer to Seat (A)**

I believe this code falls under the achievement level category 70% to 100% for this criterion due to the following reasons:

1. The program loads and displays waiting room list from CWK1;
2. A six-sided die has been used to generate the number of passengers who should move from the room to the train queue at a time.
3. Shows an error box if the queue is full;
4. The passengers can add themselves from the cwk1 without having to re-enter everything;
5. The train queue GUI is properly arranged.

**Criteria – View All Seats – GUI (V)**

I believe my code falls under the achievement level category 70% to 100% for this criterion due to the following reasons:

1. The program has visualized the train, with 42 slots and its names in it;
2. If not displays empty;
3. The gui interface is also user-friendly.

**Criteria – Delete Customer from Seat (D)**

I believe my code falls under the achievement level category 70% to 100% for this criterion due to the following reasons:

1. The data has been deleted from the queue without issues;
2. The queue is also re-ordered without an issue;
3. The deleted record is also shown.

**Criteria – Store Data to File (S)**

I believe my code falls under the achievement level category 70% to 100% for this criterion, but not 100% due to the following reasons:

1. The program stores the queue details into a text-file without an issue;
2. However, maximum marks cannot be awarded since database has not being used.

**Criteria – Load Data to File (L)**

I believe my code falls under the achievement level category 70% to 100% for this criterion, but not 100% due to the following reasons:

1. The program allows to load the booking details from the file back to the data structure without any issues.
2. However, maximum marks cannot be awarded since database has not being used.

**Criteria – Run the simulation – Run (R)**

I believe my code falls under the achievement level category 70% to 100% for this criterion due to the following reasons:

1. The program has implemented the simulation (adding time has been done in part A itself);
2. Three dice have been used to get a random time;
3. The program also shows up a gui showing all the needed data;
4. The same data has also been saved in a text file created using the current date.

**Criteria – Self Assessment, Demonstration and Code Quality**

I believe my code falls under the achievement level category 70% to 100% for this criterion due to the following reasons:

1. I have followed proper naming convections, proper indentations and have adequate comments throughout the code.

# **Technical Discussion**

For testing of this code, mainly white box testing has been used. The program has tested every aspect thinkable of to ensure a quality final product.

The main reason behind using white-box testing to test this code is because it allows to find errors and logical problems easily. Due to familiarity with the code, it has been easier to test its core functionality. White box testing does limit the testing strategy to the mind to the developer, but to ensure that the program does not leak since this has been a personal assignment, my hands have been tied to white box testing.

# **Conclusion**

I believe my code is a fully functioning code which allows a simulation to take place. Thus, the code can be used as a simulator at a train station.

Hope you enjoy the code. Thank you