Object Orientated Programming Coursework – COA256 – Train Route Planner

The aim of the coursework was to produce a train route planner system using the Java language. It was to be run in either the command prompt, or programmed to use GUI. I decided to use the GUI as I think it’s a bigger challenge and at the same time the result is a lot more visually pleasing.

How to run the code

Compile the startPlanner.java file, with the Route.java and usefulMethods.java files in the same folder.

Firstly load the program.

**Price** – to find the cost of a journey for a certain route, first click the price button. Choose the initial station and then click ‘ok’. Then choose your final station and click ‘ok’. Then choose the month that you are planning to travel and click ‘ok. Finally choose the day on which you wish to travel and click ‘ok’. A message will appear telling you the cost of your travel.

**Time** – to find the time it will take for a certain route, first click the time button. Choose the initial station and then click ‘ok’. Then choose your final station and click ‘ok’. A message will appear telling you the length of your journey.

**Route** – to check the route of your train, first click the route button. Choose the initial station and then click ‘ok’. Then choose your final station and click ‘ok’. A message will appear telling you the stops on your journey or if there are none, telling you that it is a direct route.

**Input Route** – to input a route you need to first click the admin button, select input route and click ‘ok’. Choose the initial station of the route you wish to alter and then click ‘ok’. Then choose your final station and click ‘ok’. You can then select the number of stops you wish to enter, do this and then click ‘ok’. One by one type in the stops you want to enter, click ‘ok’ after every stop.

**Save Route** – to save all inputted routes, you need to first click the admin button, select save route and click ‘ok’. You then need to type the name you wish to call the file, or choose a file to overwrite, and click save.

**Open Route** - to open a file and import all routes from it, you need to first click the admin button, select open route and click ‘ok’. You need to then select the file which contains the routes and then click ‘ok’.

Functionality working

I have been able to complete all tasks given. The program can:

* Tell you the cost of your journey along with increasing the cost if travel is on the final day of the month and also inform the user that this has happened.
* Tell you the length of your journey.
* Tell you all the stations that your train will stop at, and let admin input this information.
* Save and load information about routes.

Functionality not completed

I have not been able to specify a file type to save/open the route information as. Therefore if you try to load in a random file into the program, no error message occurs and the program just continues as normal. Also there is no catch for if when inputting a station, the user types nothing.

Design

**startPlanner**

The first thing my code does is setup the main java window. I set this to a size which nicely fits all the buttons and I made the window un-resizable. I also put in some code so the program stops running if the user exits the window. The code then adds the buttons to a panel, and the panel is placed in the frame.

It then sets up an object array called route table, where the objects are Route, something I made myself. This array contains all the information about the routes the trains take. It calls for 25 route objects to be created, one for each possible journey, and it does this using a loop.

The buttons are then registered to the ActionListener class pressed.

**pressed**

The class first creates various objects, arrays and ints needed in the code. It then identifies the button clicked but using event.getSource().

**endButton** – the end button simply ends the program being run.

**priceButton** – This firstly asks the user to choose their initial station. Then it runs the getPossibleStations method, this takes all the possible stations and removes the selected station and returns an array. The user is then asked to select their second station, from a combo box formed of this array. The user is then asked to choose their month of travel. The getPossibleDays method is then called which returns an array of the days in that month. The user is then asked to select their day of travel, from a combo box formed of this array. The program then turns the selected stations into integers. Using the getStationNo method it assigns a number to the station, 1 for London, 2 for Loughborough and so on. These integers along with the month and day selected are called into the getPrice method.

*getPrice* – this uses the station numbers to find the price. It creates a 2d array from all the prices in a regular array using a loop. This aligns the station numbers to the array of prices, so it can easily find the price by going to array[station no][station no]. It then turns the price found into a 2 decimal place double, as it is a price. It then comes to an if statement which checks if travel is on the last day of the month. If it is, it will add 10% to the price and put the price into a string, with the notification that the price is higher. If travel isn’t on the last day of the month it simply puts the price into a string. This string is then returned to the pressed class.

The string returned is notified to the user.

***Good Points – I have made it so the user is forced down a path where they cannot select the same station to travel to and from, avoiding human error. I have done this by taking their initial selection and taking it away from the second list. I have done a similar thing with the date, firstly the month is chosen and I have then given a list of days only in that month to be chosen.***

**timeButton** - This firstly asks the user to choose their initial station. Then it runs the getPossibleStations method, this takes all the possible stations and removes the selected station and returns an array. The user is then asked to select their second station, from a combo box formed of this array. The program then turns the selected stations into integers. Using the getStationNo method it assigns a number to the station, 1 for London, 2 for Loughborough and so on. These integers are then called into the getTime method.

*getTime* - this uses the station numbers to find the time. It creates a 2d array from all the times in a regular array using a loop. This aligns the station numbers to the array of times, so it can easily find the time by going to array[station no][station no]. It then uses an if statement to see if the journey is more than 60 minutes. If it isn’t it returns a string saying the length of the journey in minutes. If it is it works out how many hours and minutes the journey is and returns this in a string.

The string returned is notified to the user.

***Good Points – It doesn’t have any grammatical error when saying hours and minutes and just minutes due to the use of the if statement.***

**routeButton** - This firstly asks the user to choose their initial station. Then it runs the getPossibleStations method, this takes all the possible stations and removes the selected station and returns an array. The user is then asked to select their second station, from a combo box formed of this array. The program then turns the selected stations into integers. Using the getStationNo method it assigns a number to the station, 1 for London, 2 for Loughborough and so on. It then uses the station numbers to access the route from the route array made earlier. If calls the getRoute method of that object. If there is no route added it simply displays the default, direct route, message.

**adminButton** – this gives the user 3 options in a combo box of input route, save route and load route.

**Input route** – This firstly asks the user to choose their initial station. Then it runs the getPossibleStations method, this takes all the possible stations and removes the selected station and returns an array. The user is then asked to select their second station, from a combo box formed of this array. It creates a new route object and calls the setRoute method.

*setRoute* - this asks the user how many stops they want to enter. It then uses a loop to allow the user to enter each stop one at a time. It stores this in a string, under the value name listroute in the object.

This new route then replaces the old route object in the route array and places it in the right places by turning the selected stations into integers.

**Save route** – this opens the save dialog and saves the current route array, which contains all the route objects.

**Open route** – this opens the open dialog box and allows the user to select a file to open. Once opened it replaces all the information in the route array with the information from the file. It uses a loop to replace each route individually as the route array is final so it cannot be replaces as a whole.

**route**

Route is a class which helps the program set and show routes. It has a variable listroute, which is the list of stops on a route. This is defaulted to a direct route, so if setRoute is called it has a message to display. This listroute can then be modified by the user to change the route.

Class Diagrams

getRoute()

setRoute()

getPrice( int val, int val, String val, String val )

getTime( int val, int val )

getStationNo( String val)

getPossibleDays( String val )

getPossibleStations( String val )

private String listroute

Operations

Attributes

route

Operations

usefulmethods

final routetable[][]

Attributes

startPlanner

Tests

I tested each function of the program several times and each time the output was correct. Below are examples of my tests.

**Price #1** – I pressed the price button, and selected the journey from Derby to Middlesbrough and entered the date 19/01. The program returned a price of £19 which was the correct result.

***This test shows that getting the price for a certain route works.***

**Price #2** – I pressed the price button, and selected a journey from Derby to Middlesbrough and entered the date 31/01. The program returned a price of £20.90, and it also told me my price is higher as I am travelling on the last day of the month, which was the correct result.

***This test shows that increasing the price by 10% for last day travel, and notifying the user of this, works.***

**Time #1** – I pressed the price button, and selected a journey from Derby to Middlesbrough. The program returned a time of 59 minutes, which was the correct result.

***This test shows that getting a time for a certain route works.***

**Time #2** – I pressed the price button, and selected a journey from London to Derby. The program returned a time of 1 hour and 40 minutes, which was the correct result.

***This test shows that getting a time in hours and minutes, for a certain route works.***

**Route #**1 – Without entering any information beforehand, I pressed the route button and selected the journey from London to Loughborough. A message then told me that it is either a direct train or there are not inputted stops, which was the correct result.

***This test shows that there is a default value if no route is entered.***

**Route #2** – I clicked the admin button then chose input route. I chose the journey London to Loughborough and chose to add 3 stops. I named these stops ‘1’, ‘2’ and ‘3’. I then clicked the route button again and viewed this route. A message came up telling me my train stops at 1, 2 and 3. I then chose another route and the default message was displayed.

***This test shows that I can input stops for a certain route and view it successfully, without affecting all other routes.***

**Save and Open Route** – After performing the route#2 test, I clicked the admin button and chose save route. I then typed in a file name and saved it. I then exited the program and loaded it again. I viewed the route London to Loughborough and the default message was displayed. I then loaded in the file saved earlier. I then again viewed the London to Loughborough route and it told me that my train would stop at 1, 2 and 3. All other route displayed the default message. This is the correct result.

***This test shows that I can save and load the route that I enter into the program.***

Along with the tests of these functions, I have tested my program to make sure it combats any human error I can think of. Due to the restrictions I put in place, the user cannot select the same station to travel to and from, or choose a day in a month which does not exist eg. 31st February.

Evaluation

I think that my program has been designed well and does everything specified. It is simple and easy to use and has also been nicely presented using GUI. I have used loops, if statements and other techniques to save writing a lot of code.

I think that its strong points are that it reduces human error by sending users on some forced paths. For example when the user selects one station for departure, the program takes this option away from the list of stations for arrival and therefore stops the user entering the same station twice. A similar thing happens when the user selects the month of travel. They cannot enter the wrong date eg. 31th Feb as the program stops this by using tailor-made lists matched to each month.

However, there are some areas which could be improved. Both combo boxes, where the stations are chose, could be on one panel and auto update when they change. Another improvement that could be made is that the amount of stations that can be entered on one route is limited to 20, code could be written so that the user could type the number of stations that could be added to a route and there it could be infinite. Also there is no catch for if when inputting a station, the user types nothing.

After evaluating my coursework I believe I had written a solid program and that it checks all the boxes that the specification laid out.