COA256 Object-oriented Programming & Algorithms
Assignment 2 Specifications

Date set: 1/5/2015 Deadline: 20/5/2015 4pm

Train route finder

You are to develop a train route finder program in Java.

The program is for a company which is in charge of five train stations in central England and users need to find journey times, routes (i.e., which stops link these five stations), and costs for journeys between the five stations. Information on routes (stops between stations) can only be added, or loaded from file, by an administrator when the program is running.

The following table provides the station names, costs and journey times for the journeys (first value cost in £, second value travel time in minutes; rows indicate the origin and columns the destinations: a journey from Leicester to Derby hence costs £13.00 while the return journey costs £13.70):

Train Stations	Leicester	Loughborough	Nottingham	Derby	York
Leicester	-	2.50/10	3.50/30	13.00/48	23.50/65
Loughborough	2.50/10	-	1.50/15	2.25/23	11.50/60
Nottingham	3.50/30	1.50/15	-	2.50/12	11.50/40
Derby	13.70/48	2.00/25	2.50/10	-	11.20/45
York	22.20/70	12.00/60	11.20/40	11.20/45	-

Your program should have an interface with seven options: Time, Price, Route, Split Ticket, Sort Routes, Admin and End (which terminates the program).
The Admin option takes the user to another (Administrator) interface (no login/password is required for entering the Admin menu) with the following four options: Input route; Save route; Retrieve route; Exit (back to the previous menu).

Functionality for each of the menu options is explained in the following:

- Time: Provides the time of travel between two stations presented in hours and minutes.
- Price: The cost of travelling between any two of the five stations. Note, that if the user is travelling on the last day of the month, the fare should be reduced by 10% (and a message displayed to say that this has been done). This is a discount scheme offered by the train company. The system will therefore need to ask the user the date on which they are travelling. While dates should be entered as numbers (e.g. 26/5/2015), the output should be more verbose with the name of the month written out (e.g. 26 May 2015).

- Route: Displays a list of stops that are made between two stations. These stops are added by using the Admin option menu or retrieving the information from a previously saved data file. When the program is first run, it should check for a default data file containing route information; if such a file does not exist, there will not be any stops between stations.
- Split Ticket: As you can see from the table, e.g. going from Leicester to Loughborough and from there onwards to Derby is cheaper than going from Leicester to Derby directly. This option should hence allow the user to search for such cheaper journeys by splitting it into two separate journeys with a combined lower price. Note, that in here you are expected to arrive at such splits programmatically and not by reading off such splits from the provided table and hardcoding them in the application.
- Sort Routes: Should list all (20) routes sorted by a criterion specified by the user. The program should allow sorting by price, time, and number of stops en route.
- Input route: Allows the administrator to add as many stops as they like between any two stations. For example, one might add 'Barrow on Soar', 'Rothley', and 'Quorn' as stops between Leicester and Loughborough. You could ask the user how many stops they wish to add before they enter each of the stops in turn.
- Save route: Save all the information on journey stops (i.e. all stops for all routes) to a file in a format of your choosing (you do not need to save costs and journey times as they are fixed and we assume that although we can add stops this does not affect the overall journey time). The Admin should be given the option to save this information either in the default data file used by the program or in another route information file.
- Retrieve route: Read in all the information on journey stops from a user selected file (a file saved using the previous option). This should overwrite any existing data in the program when it is running.

Note that the requirements do not specify what type of "interface" you have to provide. A text-based interface is hence sufficient, though if you wish to provide a graphical interface (using the swing classes) you can also do so.

Make sure that your program is able to cope appropriately with unexpected input (e.g. a user trying to find the cost of travel from station A to station A, or entering an invalid date), and that all the output is clear and neatly formatted.

Also, ensure that your code is neatly laid out, that you use meaningful names, and that you include appropriate comments in the code.

Documentation

Your coursework should be submitted with a report, which should (only) contain the following sections:

- Design: to contain class diagram(s) and explanations on how/why you have designed the classes that you have in your solution. This section should also include information on the file format you have adopted for saving/retrieving route information. Any other information/assumptions that you think useful can also go in here.
- Functionality: an indication of which of the required functionality is working and which was not completed; see the Appendix for the table to copy in here.
- Test plans: to contain evidence (e.g. in form of screen shots with accompanying explanations) that your program is indeed working correctly and the tests that you have run to determine that.

Hand-in

Hand-in is electronically on learn. You should submit a single ZIP archive that contains all the files of your submission. The filename of the files must be *lastname_firstname.zip* (with *lastname* and *firstname* obviously being replaced by your family and first names). Note, that only ZIP files will be accepted and that upload of other formats will result in loss of marks.

Your submitted file should contain the following:

- A filled-in coursework cover sheet.
- Your report (either in PDF or Word format; no other formats will be accepted.
- All the source files of your program.
- NetBeans projects files for your program.
- The compiled program file(s).
- One or more route data files used for testing the program.
- Any other material you might wish to include.

(The easiest way of getting all the source and project files as well as the compiled program into an archive is to build your project on one of the lab machines and then zip your project folder.)

Marking

Marks are split according to the following scheme:

Program design 10%
Classes, class functionality, etc. as evidenced by your documentation in the report.

Functionality 65%
The more functionality you add to the program the higher your mark. Begin by getting the easier parts working and add the more complex ones later.

Testing and error handling 10%
Evidence of testing, test data, error handling and error recovery, etc.

Output, code layout and documentation 15%
Clear output formatting, code layout (indentations etc.), *useful* comments in code, documentation (report) etc.

Appendix

The following is the table that, filled in, you have to copy into the Functionality part of your report.

Functionality	**Y (complete) P (partial) N (none)**	**Comments (e.g. more details on what is not working etc.)**
Search for price		
Search for travel time		
Display route		
Split ticket		
Sort route data		
Input route		
Load route (from file)		
Save route (to file)		
Handling dates correctly		
Error handling		