CSSE2002 / CSSE7023 **Semester 1, 2014**

Assignment 3

Goal: The goal of this assignment is to gain practical experience with implementing GUIs using the Model-View-Controller design pattern.

Due date: The assignment is due at **3pm on Friday 6 June**. Late assignments will lose 20% of the total mark immediately, and a further 20% of the total mark for each day late. Only extensions on Medical Grounds or Exceptional Circumstances will be considered, and in those cases students need to submit an application for extension of progressive assessment form (http://www.uq.edu.au/myadvisor/forms/exams/progressive-assessment-extension.pdf) to the lecturer (email is acceptable) or the ITEE Enquiries desk on Level 4 of GPSouth Building) prior to the assignment deadline.

Problem description: In this assignment you will develop a GUI for finding journeys in a transport (i.e. bus) network using the Model-View-Controller design pattern. This extends the work you have done in Assignments 1 and 2.

Exactly what the GUI looks like is up to you, but in order to meet testing requirements, it must be capable of performing the following tasks:

- When the journey planner program is executed (by running the main method in `JourneyPlanner.java`), the program should load the timetable defined in `timetable.txt` using the `read` method from `TimetableReader`. An appropriate error message should be displayed if there is an error reading from the input file or there is an error with the input format of the file (i.e. if `TimetableReader.read` throws an `IOException` or a `FormatException`). The program is not required to perform any more functionality if the file cannot be read.

  You may assume that the timetable, if properly read, satisfies the preconditions stipulated in `JourneyFinder.findJourney`.

- Assuming that the timetable can be read, the user should be able to use the program to search the timetable for a journey between any two stations, departing no earlier than a given time. The user should have the freedom to specify the stations and the time. Having initiated a search request, the program should search for a journey that departs the given start station no earlier than the given time, and arrives at the given end station no later than any other such journey. If such a journey exists it should be displayed to the user in a readable format, otherwise the user should be warned that no such journey exists. (You should be using `JourneyFinder.findJourney` to perform the search.)

  You don't have to use the `toString` method of the `Journey` class to display the journey, but your representation of the journey should include the total travel time of the journey and the number of transfers in the journey. For each leg of the journey the user should be able to clearly discern the start time, end time, start station, end station and route name of the leg.

  Users should be able to perform as many searches as they like, one at a time – users don't need to be able to perform multiple searches at the same time, or view the results to multiple searches at the same time etc.

- If a user-initiated search for a journey has been successful, the user should have the option of changing the returned journey by departing one of the *interchange* stations in the journey – a

station where the user must transfer from one service to another – at some time after the current journey departs from that station. If the user requests that the journey is changed by departing an interchange station no earlier than a given time (after the current departure time from that station), then, if possible, the journey should be changed by *replacing the part of the journey from the interchange station to the end station* by a journey between those two stations that departs the interchange station no earlier than the specified time, and arrives at the end station of the journey no later than any other such journey (e.g. use method `findJourney` again to get this part of the journey). If no such journey exists, then the user should be informed and the original journey should not be changed.

(Note that no method for replacing part of a journey exists in the Journey class and so you might need to just create a new journey and populate it with the legs that you want it to have.)

- Your program should be robust in the sense that incorrect user inputs should not cause the system to fail. Appropriate error messages should be displayed to the user if they enter incorrect inputs.

For example, if the user searches for a journey from `UQ Lakes` to `Wesley` departing no earlier than time `10` in the timetable defined in the file `timetable.txt` by:

```
66
UQ Lakes, Southbank, Cultural Centre, City, Roma Street
8  12 20 24 27
20 23 30 34 40

440
City, Wesley, Toowong, Kenmore
25 30 35 50
35 40 45 60
45 50 55 70
55 60 65 80
```

then the journey with string representation

```
Total travel time: 20 Transfers: 1
20 - 34: catch route 66 from UQ Lakes to City
35 - 40: catch route 440 from City to Wesley
```

should be displayed in your chosen format. If the user then requests that the journey be changed by departing the `City` no earlier than time `40` then the journey should be updated to

```
Total travel time: 30 Transfers: 1
20 - 34: catch route 66 from UQ Lakes to City
45 - 50: catch route 440 from City to Wesley
```

and again displayed to the user.

Task: Using the MVC architecture, you must implement `JourneyPlanner.java` by completing the skeletons of the three classes: `PlannerModel.java` , `PlannerView.java` and `PlannerController.java` that are available in the zip files that accompanies this assignment. (Don't change any classes other than `PlannerModel.java` , `PlannerView.java` and `PlannerController.java` since we will test your code with the original versions of those other files.)

You should design your interface so that it is legible and intuitive to use. The purpose of the task that the interface is designed to perform should be clear and appropriate. It must be able to be used to perform the tasks as described above.

As in Assignment 1 and 2, you must implement `PlannerModel.java`, `PlannerView.java` and `PlannerController.java` as if other programmers were, at the same time, implementing the classes that instantiate them and call their methods. Hence:

- Don't change the class names, specifications, or alter the method names, parameter types, return types, exceptions thrown or the packages to which the files belong.
- You are encouraged to use Java 7 SE classes, but no third party libraries should be used. (It is not necessary, and makes marking hard.)
- Don't write any code that is operating-system specific (e.g. by hard-coding in newline characters etc.), since we will batch test your code on a Unix machine.
- Your source file should be written using ASCII characters only.
- You may define your own private or public variables or methods in the classes `PlannerModel.java`, `PlannerView.java` and `PlannerController.java` (these should be documented, of course).

Implement the classes as if other programmers are going to be using and maintaining them. Hence:

- Your code should follow accepted Java naming conventions, be consistently indented, readable, and use embedded whitespace consistently. Line length should not be over 80 characters. (Hint: if you are using Eclipse you might want to consider getting it to automatically format your code.)
- Your code should use private methods and private instance variables and other means to hide implementation details and protect invariants where appropriate.
- Methods, fields and local variables (except for-loop variables) should have appropriate comments. Comments should also be used to describe any particularly tricky sections of code. However, you should also strive to make your code understandable without reference to comments; e.g. by choosing sensible method and variable names, and by coding in a straightforward way.
- Allowable values of instance variables must be specified using a class invariant when appropriate.
- You should break the program up into logical components using MVC architecture.
- The methods that you have to write must be decomposed into a clear and not overly complicated solution, using private methods to prevent any individual method from doing too much.

Hints: You should watch the newsgroup (uq.itee.csse2002), and the announcements page on the Blackboard, closely – these have lots of useful clarifications, updates to materials, and often hints, from the course coordinator, the tutors, and other students.

The assignment requires the use of a number of components from the Swing library. You should consult the documentation for these classes in order to find out how they work, as well as asking questions on the newsgroups and asking tutors and the course coordinator.

User interface design, especially layout, is much easier if you start by drawing what you want the interface to look like. You're layout does not need to be fancy (just legible, intuitive etc. as above),

and we don't expect you to use a layout tool to create it – if you do then your code quality might not be very good. You'll have to have a look at the java libraries (http://docs.oracle.com/javase/7/docs/technotes/guides/swing/index.html) to work out what widgets that you'd like to use, and what layout options (e.g. http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html) are available.

You can read about the MVC design pattern in the lecture material from week 8 – the Calculator code example might be a useful reference.

Submission: Submit your files **`PlannerModel.java`**, **`PlannerView.java`** and **`PlannerController.java`**, electronically using Blackboard according to the exact instructions on the Blackboard website:

https://learn.uq.edu.au/

You can submit your assignment multiple times before the assignment deadline but only the last submission will be saved by the system and marked. Only submit the files listed above.

You are responsible for ensuring that you have submitted the files that you intended to submit in the way that we have requested them. You will be marked on the files that you submitted and not on those that you intended to submit. Only files that are submitted according to the instructions on Blackboard will be marked.

Evaluation: Your assignment will be given a mark out of 15 according to the following marking criteria.

Manual Testing of the GUI (6 marks)

We will manually test the expected functionality (as described in this handout) of your GUI by attempting to perform a number of scenarios.

Full marks will be given if your interface can be reasonably used to perform all of the defined scenarios correctly. Part marks will be given based on the number of scenarios that your GUI is able to perform correctly. Work with little or no academic merit will receive 0 marks.

Note: code submitted with compilation errors will result in zero marks in this section. We will execute your code by running the main method defined in `JourneyPlanner.java`.

User interface design (3 marks)

- Interface is legible and intuitive to use. The purpose of the task that the interface is designed to perform in clear and appropriate. The interface can be easily used to perform the tasks as defined in this handout. No additional and unnecessary features.
  3 marks
- Minor problems, e.g., some aspect of the interface is not able to be well-discerned
  2 marks
- Major problems, e.g. the purpose of the task that the interface is designed to perform is not clear and appropriate, or the interface cannot be used to perform most of the tasks as defined in this handout. 1 mark
- Work with little or no academic merit 0 marks

Note: code submitted with compilation errors will result in zero marks in this section. We will execute your code by running the main method defined in `JourneyPlanner.java`.

Code quality (6 marks)

• Code that is clearly written and commented, and satisfies the specifications and requirements	6 marks
• Minor problems, e.g., lack of commenting	4-5 marks
• Major problems, e.g., code that does not satisfy the specification or requirements, or is too complex, or is too difficult to read or understand.	1-3 marks
• Work with little or no academic merit	0 marks

Note: you will lose marks for code quality for:

- breaking java naming conventions or not choosing sensible names for variables;
- inconsistent indentation and / or embedded white-space or laying your code out in a way that makes it hard to read;
- having lines which are excessively long (lines over 80 characters long are not supported by some printers, and are problematic on small screens);
- for not using private methods and private instance variables and other means to hide implementation details and protect invariants where appropriate
- not having appropriate comments for classes, methods, fields and local variables (except for-loop variables), or tricky sections of code;
- inappropriate structuring: Failure to break the program up into logical components using MVC architecture
- monolithic methods: if methods get long, you must find a way to break them into smaller, more understandable methods using procedural abstraction.
- incomplete, incorrect or overly complex code, or code that is hard to understand.

School Policy on Student Misconduct: You are required to read and understand the School Statement on Misconduct, available on the School's website at:

http://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct

This is an individual assignment. If you are found guilty of misconduct (plagiarism or collusion) then penalties will be applied.

If you are under pressure to meet the assignment deadline, contact the course coordinator **as soon as possible**.