CSSE2002 / CSSE7023 **Semester 1, 2015**

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 **4pm on Wednesday 3 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 helping festival-goers plan their day at a music festival 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 festival planner program is executed (by running the main method in `FestivalPlanner.java`), the program should
  - load the shuttle timetable for the festival defined in the file `timetable.txt` using the `read` method from `ScheduleReader`.
  - load the festival line-up defined in the file `lineUp.txt` using the `read` method from `LineUpReader`.

  An appropriate error message should be clearly displayed (in the graphical user interface) if there is an error reading from either input file or there is an error with the input format of either file (i.e. if `ScheduleReader.read` or `LineUpReader.read` throws an `IOException` or a `FormatException`).

  The error message displayed to the user should clearly identify the reason for the error. That is, it should identify which input file (the timetable or the line-up) could not be loaded, and why. The reason should include whether or not the load error was due to an input/output error reading from the file, or because of an error with the input format of the file, and it should include the detail message of the exception thrown to help the user track down the exact reason for the problem. (If there is an error loading more than one input file then it is sufficient to report the error message for only one of these two problems.)

  If either the timetable or line-up could not be loaded then the program, after informing the user of the problem, is not obliged to perform the remainder of the tasks specified here. It should either exit gracefully or allow the user to close the program without being able to perform any other functions.

  Note that sample files called `timetable.txt` and `lineUp.txt` are included in the assignment zip file, but that your code should not be hard-coded to only handle the contents of those input files. Also note you don't need to check the timetable and the line-up for compatibility, because any timetable is implicitly compatible with any line-up because it doesn't matter if there is a service to or from a venue with no events, or if there is an event at a venue without any services etc.

Assuming that the timetable and line-up for the festival can be loaded, the user should be able to use the program to create a compatible day plan for the festival. Recall from assignment two that a day plan for a festival is a list of events ordered (smallest to largest) by session number. The day plan under construction should be displayed to the user in a readable format that includes each of the events in the plan (ordered by session number). For each event in the plan the user should be able to discern the act, session and venue of the event.

- When the program starts the user's proposed plan should originally be empty, and it should be evident to the user that their plan is initially empty.

- At any point, the user should be able to select an event from the line-up of the festival (if there are any events in the line-up of the festival), and attempt to add it to their plan. If either
  (i) the event already exists in the plan, or
  (ii) there is already an event in the plan scheduled for the same session, or
  (iii) it is not possible to reach the new event from the *previous event*[1] in the plan (if there is a previous event) or
  (iv) it is not possible to get from the new event to the *next event* in the plan (if there is a next event)
  then the new event is not compatible with the plan, and the user should be notified of the incompatibility and the reason for the incompatibility (one of the reasons (i)-(iv) above), and the plan should not be modified. Otherwise, if the new event is compatible with the plan, then it should be added to the plan, and the new plan should be displayed in a readable format to the user. If there is more than one reason for an incompatibility, then the user should only be informed of the first reason from (i)-(iv) above that applies.

  To determine if (iii) or (iv) holds you should use the public `canReach(Event source, Event destination)` method in the `DayPlanner` class. (Note that this extra method has been added to the `DayPlanner` class to help you with this assignment.) Reachability is defined with respect to the shuttle timetable of the festival (as read in when the program starts).

- At any point when the plan has one or more events, the user should be able select an event from their plan and remove it. After a removal, the new plan should be clearly displayed to the user.

- 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.

Task: Using the MVC architecture, you must implement `FestivalPlanner.java` in the `festival.gui` package 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.

---

[1] Given an `Event e`, the *previous event* in the plan, is the event in the plan with the largest session number less than `e`. (Such an event may not exist.) Similarly the *next event* in the plan, is the event in plan with the smallest session number greater than `e` (and such an event may not exist either).

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 8 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 piazza forum and the announcements page on the Blackboard closely – these sites 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 piazza 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 (https://docs.oracle.com/javase/8/docs/technotes/guides/swing/index.html) to work out what widgets (https://docs.oracle.com/javase/tutorial/uiswing/components/index.html) that you'd like to use, and

what layout options (e.g. https://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 will 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 reasonably be 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. A Java 8 compiler will be used to test code. We will execute your code by running the main method defined in `FestivalPlanner.java`.

Usability (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 or the interface can't be used to perform all of the tasks defined in the handout (if some aspect isn't functional, then we can't check how useable it is). 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. A Java 8 compiler will be used to test code. We will execute your code by running the main method defined in `FestivalPlanner.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**.