CAB302 Software Development Assignment 2

Test-Driven Development and Graphical User Interface Programming
Part 2: GUI Development
Semester 1, 2016

Due date: Thursday, 2 June 2016 (Week 13)
Weighting: 35% in total (17.5% for Part 1, 17.5% for Part 2)
Assessment type: Group assignment, working in pairs.
Version: 0.9: Monday, 16 May 2016 9:00PM.

Introduction:

This second part of the specification is here to provide some guidance on the development of the GUI application. In part 1, you were required to implement key classes for a system to simulate the operations of an airline. Please note that this task concerns *simulation of the bookings of an airline* – not management of that operation.

The GUI task here is thus simpler than it might be, and is concerned far more with reporting than it is with any sort of physical display. A physical display would be fun, but there is enough complexity here in the logging and display using a graphing library. I will provide you with example code that produces a simple frame, and examples of the use of the JFreeChart library.

We begin with a quick summary of the tasks to complete this assignment.

Tasks to Complete the Assignment:

Most of the development tasks were well specified in the first part of the specification. In summary, and in a sensible order, the tasks are as follows:

1. Develop classes in the `Aircraft` and `Passenger` hierarchies. .
2. Develop a JUnit test suite for the most important classes – as described in the earlier specification. It is expected that these will be comprehensive but not ridiculous – cover the basics, with normal and valid boundary cases, and the exceptions thrown.

In this document, we add some additional tasks focused on the construction of the GUI:

3. Develop a Swing application using a text area, drawing on the facilities from the first part of the assignment, and from the simulator code which constitutes the major portion of the part 2 code drop. This will be built around the class `GUISimulator.java`, (which probably could have been called something else, but we will live). The goal here will be to organize the text area to receive the output from the logging facilities provided for you. The idea is that this class should provide the frame that should then be invoked from the main program that exists in the `SimulationRunner` class. Obviously, this will have to be adapted significantly (see below).

4. Once you have this basic structure, develop your test inputs and confirm that the logging behaves as expected – before moving on to the charting.
5. Next, consider the charting needed for the graphics display (see below), and embed it in a class called `ChartPanel.java`. This class should also go in the `asgn2Simulators` package.
6. Finally, you should finish the testing of the overall application – this requires a fair bit of work, and is described below. It is permissible to do this manually, or via the use of an automated GUI testing framework. This is described in more detail later in the document.

GUI Development:

For GUIs of this limited sophistication, the structure is pretty clear cut. Work mainly through the Class constructor and the `actionPerformed` methods as the main public API for the classes you develop. Inevitably you will use some of the methods inherited from JPanel and JFrame and there will be an abundance of private helper methods.

The structure here also requires that you collect parameters from the GUI and pass those back to wherever you are creating the `Simulator` objects. This suggests that a fair bit of the work which presently sits in the main program of the `SimulationRunner` class is going to have to sit somewhere in the `GUISimulator`. Play with the design until it makes sense. If it feels clumsy, try again until you have something reasonable. You have some flexibility here.

Remember, somehow, you are going to have to get the data from the model objects – and so you need to consider this in the design of your panels and frame. You must subclass the frame and panels as otherwise you have no control over the parameter lists. I have been careful to provide you with many methods that allow you to source application data for logging and display. The actual design is up to you, provided that you satisfy the requirements laid out in part 1 – accepting input, running the simulation, and viewing the results.

The essential steps in the use of the GUI are as follows:

1. User enters - actually adjusts (see below) - parameters through text fields or other mechanisms.
2. [Invalid parameters should be trapped and the user advised of the error – either directly via the text area, or through some colour or other signal.]
3. The user should then run the simulation. My view is that the user should be offered the choice of the text or charting view, but if you can work out a way to have both without making it look silly, I am happy for you to consider it. If there is a mix of text area and graphics, then the user should have the option of switching between them. More details on these requirements are given below.

Notes:

1. The logging methods are provided to you and these should be used as a model for some, though not all, of the GUI based display. You should, however, continue to maintain the file based log.
2. The Log as set up is not suitable for direct use in the text area – it does the writing directly to a file, and does not return a String that can be readily consumed by a `JTextArea`. So, you will need to create a method in the GUISimulator class which does some of the same job, taking the result from the status methods on the simulator class.

3. The status methods on the simulator class come in two entirely different flavours. Some return a string and are suitable for text based summaries. Others are numeric and will be very useful as source material for the charting aspects of the requirements.
4. It is not necessary in the GUI environment to log the parameters as these are clearly on display in the data entry fields. Obviously, you should populate those fields initially with the values that have come from the main program – these will be either the defaults as set in `Constants.java`, or values as set from the command line arguments.
5. The final dump of everything is excessive in the GUI environment, and best left to the log, so you should conclude with the last entry from the status.
6. My expectation in respect of charting is that we should see a daily record of the actual passenger numbers over the period from `FIRST_FLIGHT` through to the end of the simulation at `DURATION`. There should be multiple series visible:
   a. The totals for each day in First, Business, Premium and Economy;
   b. The total number of passengers flown that day
   c. The total number of seats that day that remain unfilled.
7. We should also be able to select - available at the end of the simulation – a chart which shows some aspect of the queue and refused bookings for the system. Here you have the choice of:
   a. Line graphs as above which show the evolution of the queue size and the number of refused passengers over the entire simulation.
   b. A bar graph which shows 3 bars: the maximum queue size used during the simulation (i.e. the largest number of elements that were in the queue at any stage), the number of refused passengers at the end of the simulation, and the total daily capacity of the aircraft.
8. There is a summary series of Bookings objects available to you from the Simulator class which will help enormously with the plots in point 6 above.
9. The GUI should guide the user by disabling, as appropriate, controls that are not available at particular stages of the process.

Parameters:

The constants class provides distinct parameter groups. The capacities of the aircraft – the overall capacity and the capacity of the given fare classes – are fixed for the whole assignment. We will not be adjusting the duration parameters, but we will be considering changes to the probabilities. We consider these in turn as command line arguments and GUI data entry fields.

Group 1: Don't even think about changing these or making them command line arguments. They are there for all time, for all simulations. We are done here.

```
//Basic simulation time parameters - unchangeable
//All times are given in calendar days
public static final int WEEK = 7;
public static final int FIRST_FLIGHT = 3*WEEK;
public static final int MAX_BOOKING_PERIOD = 6*WEEK;
public static final int CANCELLATION_PERIOD = 1*WEEK;
public static final int MAX_QUEUING_PERIOD = 1*WEEK;
public static final int DURATION = 18*WEEK;
```

Group 2: The probabilities – may change often. I would advise that you keep the SD roughly at that fraction of the mean for the durations.

```
//RNG and Probs - assume mean daily bookings around total capacity
public static final int DEFAULT_SEED = 100;
public static final int DEFAULT_MAX_QUEUE_SIZE = 500;
public static final double DEFAULT_DAILY_BOOKING_MEAN = 1300.0;
```

```
	public static final double DEFAULT_DAILY_BOOKING_SD = 
0.33*Constants.DEFAULT_DAILY_BOOKING_MEAN;
	public static final int MINIMUM_BOOKINGS = 300;
	public static final double DEFAULT_FIRST_PROB = 0.03;
	public static final double DEFAULT_BUSINESS_PROB = 0.14;
	public static final double DEFAULT_PREMIUM_PROB = 0.13;
	public static final double DEFAULT_ECONOMY_PROB = 0.70;
		public static final double DEFAULT_CANCELLATION_PROB = 0.1;
```

Group 1 is not considered further. For group 2, the command line program provides you with the ability to set them directly, but on an all or nothing basis.

Setting up the GUI application:

- Adapt my main program to give the option of the text output (the existing command line application) or to run your GUI. This will require a change to the number of expected arguments and to the usage message that comes out if the wrong number of arguments is used.
- If no command line arguments given, we just use the defaults. Otherwise, there must be enough for the `Simulator` constructor
- You may find the java tutorial helpful again:
  http://docs.oracle.com/javase/tutorial/essential/environment/cmdLineArgs.html

Then, once you start working with the GUI, group 2 will need to be reflected in the UI:

- If there are no command line arguments, populate the text fields (or other widgets) for the simulation parameters with the defaults.
- If we have the correct number of command line arguments, populate the text fields (or other widgets) for the simulation parameters with these new values.
- In either case, we give the user the chance to make adjustments. The user *then* must start the simulation running (probably with a start button).

There are arguments checks to be performed on the `Simulator`, and a range of exceptions may emerge during the course of the simulation. Optionally, you may deal with invalid data by an error message via the text area or modal dialog, or by:

1. Handling appropriate text change events in the text entry box and prevent the user launching the simulation until the input is valid
2. Disabling and/or greying out as appropriate when input or the display option is invalid.

Charting:

Those seeking full marks for the assignment must produce a chart based report using JFreeChart (see subsequent guide). You should have access to any `java.awt colour`. These are suggestions – they work ok, but you might like to try others.

Chart 1: Progress:
X-axis:
time – as an integer

Plotted Series:
Black – First
Blue – Business
Cyan – Premium
Gray – Economy
Green – Total
Red – Seats available

Chart 2: Summary (at end of simulation):
A final chart showing the queue and the refused passengers. As noted above, there are two alternatives:

- A line chart very similar to chart 1, but running from the start of the simulation until the end, showing the queue size each day and the number of refused passengers at the end of each day
- Or a bar graph showing the largest queue size achieved during the simulation – i.e. the maximum number of Passengers in the queue at any one time; the number of Refused passengers at the end of the simulation; and the total daily capacity of the 3 aircraft.

Note here that the number of refused passengers is a cumulative count, and the others are not.

The colour scheme is again up to you, but some suggestions as follows:

Line chart:
X-axis:
time – as an integer

Plotted Series:
Black – Queue Size
Red – Passengers Refused

Bar Chart:
X-axis:
Bar labels

Bars:
Black – Queue Size
Red – Passengers Refused
Green – Daily Passenger capacity.

GUI Testing:

Both of you should be responsible for the look and feel of the interface, but testing requires that you work through some scenarios, and it is best if you can try to test the logic the other person coded.

Testing of the GUI will be based on a series of usage scenarios and the data associated with them. These will include a few standard examples that I will supply later, but for now, explore:

- High cancellation probability (not ridiculous, 0.4 or 0.5 makes it tough enough)
- No queue
- Higher probability for First and Business (with obviously lowered economy)
- Higher and lower daily passenger demand.

In the final submission, you will document the scenario and the data used. You may demonstrate that your system performed correctly in one of two ways:

- Through a series of screenshots presented as a pdf or PPTX slide deck.
- OR (for enthusiasts) using an automated GUI testing framework such as Abbot. We will discuss this further before you have to make a choice. We must caution that even if you use an automated GUI framework, there will be issues inherent in using this for the charting environments.

Here is a formal version of a GUI test scenario from an earlier assignment, which involved a locomotive system [PAG means Perform a Gesture, which is a neutral way of saying do something without making it a button or some other specific widget]. The plain font is what you do. The italicised text is what the system does.

Script 1: Build a Train

1. PAG to add a Locomotive of class 4D of weight 180 tonnes
2. *The System display shows the locomotive in the train configuration, indicates that the train is not overloaded, and shows a capacity of zero passengers and that the train is full.*
3. PAG to add a passenger carriage of weight 80 tonnes and capacity 50
4. *The System display shows the locomotive and a single passenger carriage in the train configuration, indicates that the train is not overloaded, shows a remaining capacity of 50 passengers, occupancy of 0/50 in carriage 1, and indicates that the train is not full.*
5. PAG to add a passenger carriage of weight 80 tonnes and capacity 50
6. *The System display shows the locomotive and two passenger carriages in the train configuration, indicates that the train is not overloaded, shows a remaining capacity of 100 passengers, occupancy of 0/50 in carriage 1 and 2, and indicates that the train is not full.*

In our case, we follow this style, even if the test cases end up being simpler.

Assessment

GUI Design:

We have previously noted that GUI design need not be world class, but that it should not be an abomination either. Some basic principles follow, and these also are clear in the CRA. Your GUI should:

- Be well laid out with related information and functional widgets grouped together;
- Have a good alignment between the display widgets chosen and the type of information to be displayed;
- Automatically update information displays affected by other operations;
- Not appear cluttered – Google's is, or at least used to be, an example of a clean interface.

Students are permitted to use a GUI Builder such as the Visual Editor for Eclipse provided that this is explicitly declared in the submission, and provided they understand that a higher standard of GUI design will be expected for the same mark level. As always, students are permitted to work in environments other than Eclipse but the final product *must* be submitted as a working Eclipse archive. We will not under any circumstances work through a cumbersome import process for an assignment.

If you wish to use other libraries in addition to `swing` for the GUI display please contact us directly. We will look at this on a case by case basis, but if we say yes the expectations will follow the same path as for the GUI builder, with a higher standard expected for the same mark level.

Code Quality:
Whilst GUI code can at times be very ugly, there is no excuse for sloppy coding style and documentation. The standard of elegance for GUI code is lower than for other contexts, but there are still some violations that are plainly not acceptable. To make your code more readable, try:

- Some terse single line comments above a block of code - highlighting material that is less than obvious;
- Lots of private methods to make sense of lots of widget creation and organization. The refactoring functions on the right click menu are your friends. Get to know them.
- As always, we recommend following a recognised coding convention, such as that described in the *Code Conventions for the Java Programming Language*[1].

Source Control:
As for part 1, we expect that you will use a recognized source control program to manage the codebase for this assignment, and that your logs will be submitted as part of the assignment. Please do not fake your logs to make them nicer – unless you do your first commit some time the night before the assignment is due, you will almost certainly get a decent mark for this. For most of you, this is the first time you have used source control in any decent way, and so we are really just looking to see that you have given it a proper go. We will indicate our submission requirements for these in the submission doc.

[1] http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

[Similar text to part 1]
Academic Integrity
Please read and follow the guidelines in QUT's *Academic Integrity Kit*, which is available from the Blackboard site on the Assessment page. Programs submitted for this assignment will be analysed by the MoSS (Measure of Software Similarity) plagiarism detection system (http://theory.stanford.edu/~aiken/moss/).

Assessment
Part 1 of submitted assignments will be tested automatically, so you must adhere precisely to the specifications in these instructions and on Blackboard. Your program code classes will be unit tested against our own test suite to ensure that they have the necessary functionality. Your unit test classes will be exercised on defective programs to ensure that they adequately detect programming errors. As before, we will supply a testing program to enable you to make sure that your code is consistent with the spec. The precise assessment criteria for the assignment are supplied in a separate document.
Part 2 of submitted assignments will be assessed based on the functionality of your Graphical User Interface. Further details of assessment criteria for Part 2 may be found in the released CRA).

Submitting Your Assignment
Full details of required file formats for submissions will appear on Blackboard near the deadline. You must submit your solution before midnight (actually 11:59) on the due date to ensure that your assignment is accepted. You should take into account the fact that the network might be slow or temporarily unavailable when you try to submit. **Network problems near the deadline will not be accepted as an excuse for late assignments, and QUT's assignment submission policy is now very strict.**