Programming Assessed Exercise 3

General feedback

Overall the quality of work on this exercise was again excellent. The average mark was A3, and the highest mark of A1 was obtained by 25 students. It is fair to say that this exercise was more challenging than AE2 and gave a good opportunity to gain familiarity with objects and array classes. Some general comments which apply to several students are given below, under the headings of "functionality", "design" and "documentation".

Functionality

Most people managed to read in the ClassesIn.txt and AttendancesIn.txt files in order to produce the initial timetable and attendance report. A common error was to use " " instead of "[ ]+" for the delimiter when tokenising the input files. This meant that tokens separated by multiple spaces (such as in AttendancesIn.txt) would not be handled correctly.

Generally people followed the specification and stored a fitness class starting at time t at position t-9 of the FitnessClass array. (Too many "magic numbers" were used here though, i.e., using 9 directly instead of declaring a constant called EARLIEST_START_TIME or something similar.) This meant that there could be gaps at arbitrary locations within the FitnessClass array, corresponding to null pointers, which in turn corresponded to available timetable slots.

Perhaps the most challenging part of the exercise was to sort the FitnessClass array in non-increasing order of average attendance when displaying the attendance report. The main FitnessClass array (used to store the fitness classes ordered on start time as described above) could not be sorted directly for two reasons:

(i) Arrays.sort would encounter null pointers when trying to compare two FitnessClass objects;
(ii) the resulting array would in general no longer satisfy the property that a fitness class starting at time t would be stored at position t-9.

To get round these problems it was necessary to take a copy of the FitnessClass array containing only the non-null FitnessClass objects stored in the original array. This should then be sorted on average attendance using a suitable compareTo method in FitnessClass.

Common errors were forgetting to ensure that a fitness class with the same ID did not already exist when adding a new fitness class, and not checking to see that the schedule had an available slot before adding a new fitness class.

Design

Generally the implemented functionality tended to be stronger than the design. Some people were putting methods in the wrong classes, or repeating code needlessly. FitnessClass and FitnessProgram are both model classes. FitnessClass was intended to hold the information specific to a single fitness class, and to contain methods to support operations to be carried out on a single class, such as setting / getting class details including the ID, class name, tutor name, start time and average attendance. FitnessProgram is an array class, and its main

instance variable was intended to be an array of FitnessClass objects. There should not be an accessor method to return this. Methods in this class would support operations to be carried out on the FitnessClass array, including adding, deleting or sorting fitness classes.

ReportFrame is a view class – some people put much of the code relating to producing the attendance report in FitnessProgram or FitnessClass, leaving ReportFrame very short. SportsCentreGUI is a view/controller class. Some people put the file input operations inside FitnessProgram, but they sit better inside the initLadiesDay and initAttendances methods within SportsCentreGUI. The neatest way to populate the FitnessClass array, for example, is to read each line at a time from ClassesIn.txt from within initLadiesDay. Each line can be passed to an addClass method within FitnessProgram. This in turn creates a FitnessClass object – to do this it is best to have a FitnessClass constructor that takes a String parameter. This can then be tokenised within the constructor to work out the start time (and other details). The addClass method can then get the start time from the FitnessClass object just created to work out where it should be stored in the array.

Another observation is that the actionPerformed methods were generally structured very well – they were short, concise and directed the program execution immediately to the relevant helper method upon pressing a given button.

Documentation

Generally the level of commenting was better than in previous exercises. Also some people were using Javadoc-style comments – although this was not required, it is nevertheless excellent practice. However on occasion, some more complex algorithms within methods could have benefited from more commenting. Also the status reports tended to include plenty of evidence of testing, though in some cases the functionality that had been reported as delivered was not consistent with the functionality encountered when executing the program at the marking stage. It is always better for the status report to be up front about any shortcomings in the program, as discrepancies suggest a lack of testing, which is then penalised.