Department of Computer Science
Com1001 CrossOver

COM1001 Crossover Assignment 1

Hand out: Tuesday, 29th October 2013
Hand in: Monday, 11th November 2013; 12.00pm
Percentage: 10% of the overall mark

Your task for this assignment is to solve a programming exercise, generate a test set for it, document the implementation and provide a description of the processes underlying these activities that you are meant to develop as a group. This assignment requires the use of the programming language Java as taught within the context of the Com1003 module. This means your code must entirely and exclusively rely on the Java concepts taught in this module between weeks 1 - 5/6 and the framework discussed in Lecture 4 (notes available from https://staffwww.dcs.shef.ac.uk/people/M.Gheorghe/campus_only/com1001/).

1. Simple functions handling student and lecturer data

Data related to *undergraduate students* (enrolled on a three year programme, BSc degree) and *staff members* (*lecturers*) of a University Department are twice per year processed – after the exam results are approved by the Faculty board - for generating student marks and lecturer workload reports requested by the Department. These data uniquely identify each *student* and *lecturer*. Every student is attending 6 modules and gets a number of credits and a mark for each of them. Every lecturer has to teach 2 modules, each is attended by the students registered with it and has a certain weight. A module is taught either for one of the two semesters or for both of them. Each module taught for one semester is assessed at the end of that semester and, after exams, the results are sent to the Faculty board. For a year-long module the results are handled at the end of semester II.

A simple prototype system will be produced by the Department of Computer Science with the help of first year students having basic knowledge of Java and attending a software engineering module. The students will be working in teams.

The prototype system consists of a main program, called below main software framework, & **4 (for teams of 5 students) or 5 (for teams of 6 students)** key functions.

1. Main software framework. This will handle a set of key functions described below. By using an input character, between "2" and "6", the corresponding function below is selected ("2" for Input data, "3" for Validation data, etc); "1" is for exit. Any of the functions identified by a character between "3" - "6" must be invoked only after using Input data. When the program finishes an "End of program" message is displayed.

Functions

2. Input data. Data defined below is read from the standard input (in parenthesis it appears the type of the corresponding value).

1.1 For each *student*

- Ucard number (integer).
- year (an integer value, between 1-3).
- For each of the 6 modules
    - module code (string of max 7 characters).
    - number of credits (integer).
    - mark (integer) – 0 must be read when there is no mark.

1.2 For each *lecturer*

- Ucard number (integer).
- For each of the 2 modules taught
    - module code (as above).
    - module weight (integer value, 2 or 4) – 4: for year 1 and year 2 modules, 2: for a year 3 module.
    - number of students registered for this module.

The *student* and *lecturer* data are uploaded into suitable data structures that are used by other functions; a message, "Student and staff data are read and uploaded", is then displayed.

3. Validation data. Check that

- year, for student data, is between 1 - 3 and module weight, for lecturer data, is either 2 or 4.
- number of credits (student data), module mark (student data) and number of registered students (lecturer data) are all positive numbers (>0).

A message, "Student and staff data are validated", is displayed when all the above conditions are verified. If at least one of them fails then an error message is displayed for each corresponding wrong value.

4. Compute student marks. For each student the year average mark is calculated. For final year students the overall mark is computed as the sum of the weights of the year average marks. These weights are computed in accordance with the following percentages: 0% for year 1, 33% for year 2 and 67% for year 3. The year average mark and the overall mark (this only for a year 3 student), are added to the current student record. The year average mark is computed based on module marks and associated number of credits. Each student record, consisting of the student data previously read (see Input data), the average mark, and overall mark for 3rd year students, is displayed.

5. Compute lecturer workload. For each lecturer the workload is calculated by considering the module workload, for each of the two modules, given by its weight multiplied by the number of registered students. Each lecturer record consists of the lecturer data previously read (see Input data), the module workload, for each of the 2 modules, and the total workload. Each lecturer record is displayed.

6. Check modules (**this is only for teams of 6 students**). Check that for every student record and every module on it, there is at least one lecturer teaching that module and for every lecturer record and every module taught, there is at least one student registered for that module. If both properties are verified a message, "Student modules match up lecturer modules", is displayed;

otherwise each module from the student records (lecturer records) which does not appear on any lecturer records (student records) is displayed.

The task for your team is to produce a Java program that implements the main software framework and the functions described above, using the coding standards mentioned in Lecture 4.

2. Testing

The software framework and each of the functions described above will be tested by a *complete test set* which is built using the ***code coverage principle*** discussed in Lecture 4. For the framework and each of the functions implemented, a test set, expressed as a set of tuples with values read from the standard input, corresponding to certain parts of the code which are tested, is built.

3. What you submit

Each team will prepare
1. A Java file containing the implementation of the framework and the above mentioned functions.
2. A word or text document consisting of the following three sections
    2.1. **Implementation section**. A description of the Java implementation (framework and functions) for the above problem.
    2.2. **Testing section**. A list of the test sets associated to the implementation.
    2.3. **Processes section**. A description of the main processes occurring in this project, consisting of:
        2.3.1. Description of how programming and testing tasks have been allocated – provide the workload (time spent on these activities and on writing parts of the documentation) for each team member.
        2.3.2 A brief description of problems encountered. Here you should mention whether there have been technical problems in understanding the tasks allocated, finding the solution, transforming the solution into Java code or ambiguities in the description of the problem. Team related issues, if any, should be reported here.
        2.3.3 A statement of the team making clear *whether the work has been equally allocated to team members or not. If not, mention how the work has been divided.*

The two files, the Java code and the document, will be called **CodeTeam**XY and **DocTeam**XY, respectively, where **XY** is the number of your team. Any other submitted document or the submission of one large file with both requested deliverables or any other format won't be accepted.

4. How you submit

Each team will submit, before or by the deadline. Any late submission will be affected by the penalties specified by the undergraduate student handbook for late hand-in (see http://www.dcs.shef.ac.uk/intranet/teaching/public/assessment/latehandin.html). Your team files must be sent by email to Marian Gheorghe, at m.gheorghe@sheffield.ac.uk. The subject of this email must be "Com1001 – TeamXY – Assignment 1", where XY is the number of your team.

Marking scheme (out 100)

1. Implementation (correctness – 45; layout, comments, coding standards - 10) – 55
2. Algorithm description (completeness, unambiguity) - 15
3. Testing (correctness, completeness) – 15
4. Process description (task allocation – 5; solving difficult tasks – 3; statement – 2; management meeting - 5) - 15