CentraleSupélec

IS1220 - Object Oriented Software Design

Tutorial 03

Paolo Ballarini

General instructions:

- If you have not done so already, create a workspace for your tutorial `IS1220/TDs`.
- Within the workspace, create a new package for this TD called `fr.ecp.IS1220.TD03/`.
- Carefully document your code (JavaDoc), i.e., explain the general idea of your algorithm, explain the relevant steps in the code implementing the algorithm, document assumptions (if any), corner cases, and error conditions.

Learning outcomes:

- Java colletions
- Abstract classes
- Methods overriding and hiding

Patient: Mary

systolic	diastolic	day
94	61	2/5/2013
97	65	3/5/2013
144	99	4/5/2013
123	88	5/5/2013
177	110	6/5/2013
145	89	7/5/2013

Patient: John

systolic	diastolic	day
88	57	15/5/2013
95	61	16/5/2013
114	80	17/5/2013
151	96	18/5/2013
176	104	19/5/2013
176	110	20/5/2013

Table 1: Pressure data for patients Mary and John

Exercise 1. Java Collections and ArrayList

A patient is told to measure her blood pressure and record the two values (systolic and diastolic) and the day of the measurement.

Q1) Implement a JAVA program for representing blood pressure information as given in the above table.

You can organise your implementation (the classes you define for your program) as you like but you are required to use an `ArrayList` for storing the blood-records for a given patient. Although Java provides you with a number of classes for representing "Date" data (e.g. `Date`, `LocalDate`, ...) you are required to implement your own version of a class called `Date` which should fit in with the sample code we provide you.
Also your classes should work with the following piece of test code described as follows:

```
public class BloodTest {
        public static void main(String[] args) {
                Patient mary = new Patient("Mary");
                mary.addBlood_measure(new Blood(94,61, new Date(2,5,2013)));
                mary.addBlood_measure(new Blood(97,65, new Date(3,5,2013)));
                mary.addBlood_measure(new Blood(144,99, new Date(4,5,2013)));
                mary.addBlood_measure(new Blood(123,88, new Date(5,5,2013)));
                mary.addBlood_measure(new Blood(177,110, new Date(6,5,2013)));
                mary.addBlood_measure(new Blood(145,89, new Date(7,5,2013)));
                printReport(mary);
        }
}
```

Q2) Having in mind that systolic values are too high if they are above 140 your implementation should provide the possibility to print a report (e.g. method `printReport`) about the recorded measurements.

The `printReport` functionality should print:

- the highest abnormal systolic blood pressures, together with the corresponding diastolic value, and the day it has been measured;
- if all systolic values were below 140, “no measurement was too high”;
- the average diastolic blood pressure;
- the list (possibly containing a single element) of maximal pressure records for a patient.

Q3) Write a small test code where you define 2 patients and set their blood pressure records according to the data given in Table 1. Display the report for both patients.

Exercise 2. Abstract classes

You are asked to realise a JAVA program that universities (like ECP) will provide to staff members (in particulars lecturers) for managing their courses. The program should allow for representing the following entities: `User`, `Lecturer`, and `Student`. The users and the lectures are distinguished. The system handles three types of students: `PhDStudent`, `MScStudent` (master students), and `UgStudents` (undergraduate). MSc and UG students take modules. For each module (which has a name and a code for example "Advanced Programming" has code "IS1220") a student will get a `finalMark` (FM), obtained as a combination of three `markSplits`: the `finalExam` (FE), the `midTermExam` (ME), and the `PracticalWork` (PW). For MSc students the finalMark is given by the following combination $FM = 0.6FE + 0.2ME + 0.2PW$, whereas of UG students $FM = 0.7FE + 0.3PW$. Furthermore the pass mark for MSc students is 10, for UG students 8.

Q1) Build classes and methods to compute for each student their final mark for a given module starting from component marks and to determine whether they have passed or failed the module. Avoid duplication of code and make use of inheritance. Your program should contain methods `finalMark()` (that returns a value of type `int`) and `hasPassed()` (that returns a value of type `boolean`).

First build a simple class hierarchy for providing functionalities for computing the final mark of students of an (unspecified) module (hence implementing methods finalMark and hasPassed discussed above). You should allow for distinguishing between MSc and UG student bearing in mind that the above described criteria apply (i.e. pass mark is 50 for MSc and 40 for UG students, whereas the final marking is computed as described before for the two types of students). In representing a student you should consider some relevant information such as: first name, last name, username, type of program (MSC or UG) and the marks splits he/she has/will receive(d) for the module.

Q2) Write a short test program in a `TestStudent` class. This class contains a `main` method, handles 3 students, and displays on screen for each student the final mark and if she/he passed the exam (use 10 has the passMark for deciding if the exam has been passed) like follows:

- student1 : type: MSC, name: John, surname: Frusciante, user=jfrusc; MEmark=12, FEmark=14, PWmark=8
- student2 : type: MSC, name: Chris, surname: Cornell, user=ccorn; MEmark=2, FEmark=13, PWmark=12
- student3 : type: UG, name: Perry, surname: Farrel, user=pfarr; MEmark=2, FEmark=13, PWmark=12

Q3) Extend the above software design by adding the following features:

- add the possibility to represent different course `Modules`. You should identify all necessary information required to be captured by a module object (e.g. its name, its code, its lecturer, the students taking the module, the weights which are used for weighting the marks for the various type of students taking the module, the pass mark, ...);
- add the possibility to represent a Lecturer (which in the computer system of the university is also a `User` and can be even a `Student`). Again you should identify all relevant information for characterising a Lecturer (like the modules he/she teaches);
- provide the `Lecturer` class with a method `finalMark(Module m)` for computing the final marks for all students enrolled in module `m` (where m a module he/she teaches). Apart from the final mark for each student this method should display if the student has passed or failed the module.

Write a simple test program where you define *(i)* a module called `Advanced Programming` with code `IS1220`, and a lecturer called `Paolo Ballarini`; *(ii)* add to the module the 3 students (2 MSC and 1 UG) described in the example above assigning them the same mark splits as before; *(iii)* eventually compute the final mark for each student of the courses taught by lecturer Paolo Ballarini.