Lab 2

[valid 2015-2016]

Starting from this week...:

- Comment and properly format the source code of your programs (otherwise: -0.5 points)
- Use the naming conventions for writing Java code. "Naming conventions make programs more understandable by making them easier to read." (otherwise: -0.5 points)
- Use the API documentation: Java Platform, Standard Edition 8 API Specification (otherwise...)

The Student-Project Allocation Problem (SPA)
(The definition of the problem is taken from: "The Student-Project Allocation Problem", by David J. Abraham, Robert W. Irving, and David F. Manlove)

An instance of SPA involves a set of **students**, **projects** and **lecturers**. Each project is offered by a unique lecturer and has a capacity constraint (an upper bound regarding how many students can be enrolled).
Each lecturer has also a capacity constraint regarding how many students is he/she willing to supervise.
A student may be enrolled in at most one project. Each student has a preferences list over the available projects that he/she finds acceptable, whilst a lecturer will normally have preferences over the students that he/she is willing to supervise.

We consider the problem of allocating students to projects based on these preference lists and capacity constraints, the so-called Student-Project Allocation problem (SPA).

Example: 7 students, 3 lecturers, 8 projects, project capacities: c(P1) = 2, c(P2) = c(P3) = ... = c(P8) = 1, lecturer capacities: c(L1) = 3, c(L2) = 2, c(L3) = 2

Student preferences	Lecturer preferences	Available projects
S1 : P1 P7	L1 : S7 S4 S1 S3 S2 S5 S6	L1 offers P1, P2, P3
S2 : P1 P2 P3 P4 P5 P6	L2 : S3 S2 S6 S7 S5	L2 offers P4, P5, P6
S3 : P2 P1 P4	L3 : S1 S7	L3 offers P7, P8
S4 : P2		
S5 : P1 P2 P3 P4		
S6 : P2 P3 P4 P5 P6		
S7 : P5 P3 P8		

The main specifications of the application are:

- (0.5p)Create an object-oriented model of the problem. You should have at least the following classes: *Student, Lecturer, Project, Problem*.
  Each class should have appropriate constructors, setters and getters. The *toString* and *equals* methods form the *Object* class must be properly overridden.
  Note that both a student and a lecturer are **persons**, having various properties specific to humans, like: name, email, etc. You should consider creating a superclass for them. This superclass should have the abstract method *isFree* (a student is free if it has no project, a lecturer is free if its capacity has not been reached).
- (0.25p) Create and print on the screen the instance of the problem described in the example.
- (1p)Implement an algorithm for allocating students to projects, subject to the preferences and capacities. The solution should be a **matching** between students and projects. Consider creating a class to describe such a matching. Analyse how "good" is this matching from the students or lecturers point of views.
  A bonus may be awarded if you implement an algorithm to generate a stable matching, as described in the article.
- (0.25p)Write doc comments in your source code and generate the class documentation using javadoc

Resources

- Tutorial: Object-Oriented Programming Concepts
- Tutorial: Classes and Objects
- Java Language Specification: Classes

Objectives

- Create a project containing multiple classes.
- Instantiate classes and manipulate objects.
- Understand the concepts of: object identity, object state, encapsulation, property, accessors/mutators.
- Override methods of the *Object* class.
- Understand uni- and bi-directional relations among objects.
- Understand the notion of multiplicity (one-to-one, one-to-many, many-to-many).
- Implement instance-level relationships among objects (association).
- Implement class-level relationships (generalization)
- Work with abstract classes.
- Get used to the naming conventions of the Java language.
- Generate documentation using javadoc.