Lab 4

[valid 2020-2021]

Starting from this week...:

- Use appropriate collections in order to represent data (otherwise: -0.5 points)
- Use Java Stream API in order to perform aggregate operations on data.

The Student / High School Admission Problem (SAP)
An instance of SAP involves a set of *students* and a set of *high schools*, each student seeking admission to one school, and each school having a number of available places (its capacity). Each student *ranks* some (acceptable) schools in strict order, and each school ranks its applicants in some order. A *matching* is a set of pairs *(student, school)* such that each student is assigned to at most one school and the capacities of the schools are not exceeded. A matching is *stable* if there is no pair (s, h) such that s is assigned to h' but s prefers h better than h' and h prefers s better than some of its assigned students. We consider the problem of creating a stable matching between students and schools.

Example: 4 students `S0,S1,S2,S3`, 3 high schools `H0,H1,H2`, `capacity(H0)=1, capacity(H1)=2, capacity(H2)=2`.

students preferences
S0: (H0, H1, H2)
S1: (H0, H1, H2)
S2: (H0, H1)
S3: (H0, H2)

schools preferences
H0: (S3, S0, S1, S2)
H1: (S0, S2, S1)
H2: (S0, S1, S3)

A solution for this example might be: `[(S0:H1),(S1:H2),(S2:H1),(S3:H0)]`

The main specifications of the application are:

Compulsory (1p)

- Create an object-oriented model of the problem. You should have at least the following classes: *Student, School* and the main class.
- Create all the objects in the example using streams.
- Create a *list* of students, using *LinkedList* implementation. Sort the students, using a *comparator*.
- Create a *set* of schools, using a *TreeSet* implementation. Make sure that *School* objects are *comparable*.
- Create two *maps* (having different implementations) describing the students and the school preferences and print them on the screen.

Optional (2p)

- Create a class that describes the problem and one that describes a solution (a matching) to this problem.
- Using Java Stream API, write queries that display the students who find acceptable a given list of schools, and the schools that have a given student as their top preference.
- Use a third-party library in order to generate random fake names for students and schools.
- Implement an algorithm for creating a matching, considering that each student has a score obtained at the evaluation exam and the schools rank students based on this score.
- Test your algorithm.

Bonus (2p)

- Consider the case in which a school can rank the students based on their specific criteria.
- Implement the Gale Shapley algorithm in order to find a **stable** matching.
- Consider the case in which preferences are not necessarily strict. Some consecutive preferences in an element's list may form a *tie*.
  For example S1: H1, [H2,H3] means that S1 prefers H1 over H2 and H3, but H2 and H3 have no precedence one over the other.
- Prove that in the case of SAP with ties, a problem may have multiple stable matchings, not all of the same size.
- Check out other examples of matching in practice.

Resources

- Slides
- Aggregate Operations
- The Java Tutorials: Collections (Know Thy Complexities!)
- Setting the class path
- Creating, Importing, and Configuring Java Projects in Netbeans
- Packaging Programs in JAR Files

Objectives

- Use various collections and polymorphic algorithms.
- Use Java Stream API.
- Use *Optional* class and *var* keyword.
- Understand the notion of CLASSPATH.
- Use external JAR libraries.
- Create a Maven Project.
- Package all project classes as an executable JAR file.