Homework 1: CIS Waitlist - Application of Stable Matching

Due by 11:59PM ET on Monday, January 27, 2019

Required Problem (30 points)

Setup and Logistics

We have put together several stub files — **click here to download them (/~cis121/current/hw/1-stable-matching/hw1-stub-files.zip)**.

3 files to submit: `Course.java`, `CourseTest.java`, and `ProcessWaitlist.java`
Please do not submit any additional files.

Be sure to follow these guidelines:

- Ensure source files are not located in any package and that none of the files have a package declaration (e.g. `package src;`).
- When submitting, submit each file individually at the root level (not contained in a folder or zip files).
- Ensure that your own test files pass and do not run into infinite loops.
- You have unlimited submissions until the deadline on the autograder marked `[compilation]`, and **two free submissions** on the autograder marked `[full]`.
- For this assignment:
  - **Number of Free Submissions**: 2
  - **Deduction Per Additional Submission**: 5 points
  - **Max Code Coverage Deduction**: up to 5 points
  - **Max Style Score Deduction**: up to 5 points
- Do not wait until the last minute to try submitting. You are responsible for ensuring your submission goes through by the deadline.
- Before asking questions on Piazza, be sure to check and keep updated with the pinned clarifications post.
- **Gradescope issues?** Be sure to check out our **Gradescope help page (/~cis121/current/gradescope_help.html)** before posting to Piazza.

Part 1: CIS Waitlist Stable Matching (30 points)

Files to submit: `Course.java`, `CourseTest.java`, and `ProcessWaitlist.java`

Background – New CIS Waitlist Algorithm

Note: this isn't actually how the CIS Waitlist system works – just a hypothetical.

Recently, the CIS department decided to employ a variant of the Gale–Shapley algorithm to assign students to CIS courses. Essentially, every student ranks as many CIS courses from the waitlist as they want, and every professor ranks (on behalf of each course) as many students as they want. The department will then run the algorithm, and match as many students as possible with **one** course, such that the matchings are stable. An example is described below.

The algorithm you learned in lecture doesn't quite support this use case, because in this case each course might have a capacity > 1, and thus two students can match to the same course. Additionally, neither the students nor the courses are obligated to include all available choices in their preference lists. Hence, we will use a variant of the algorithm described below:

```
Initially, all students and courses are free

While there is a free student s who hasn't proposed to every course:
    Choose such a student s
    Let c be the highest ranked course in s's preference list to whom s has not yet proposed to

    If s is in c's preference list:
        If c is not yet at capacity:
            then (s, c) become engaged
        Else c is at capacity and currently engaged to student(s) s_1, ..., s_n:
            If c prefers all of s_1, ..., s_n over s:
                then s remains free
            Else c prefers s to one or more of s_1, ..., s_n:
                Kick out the least preferred student, s', from s_1, ..., s_n, and s' becomes free
                (s, c) become engaged

Return the set of engaged pairs (for this assignment, you will be modifying the course list directly instead of retur
ning anything)
```

An important note: (s, c) can be a pair only if s is on c's list and c is on s's list.

A Simple Example

You may want to trace out the following example. We've provided you with this test case as well in `ProcessWaitlistTest.java`.

```
Students and their preferences, in order from most preferred to least preferred:
+---------+---------+---------+---------+---------+
|   Sam   | Brianna | Steven  |  John   |  Tommy  |
+---------+---------+---------+---------+---------+
| CIS 520 | CIS 520 | CIS 520 | CIS 502 | CIS 520 |
|         | CIS 502 | CIS 521 | CIS 520 | CIS 502 |
|         |         | CIS 502 | CIS 521 | CIS 521 |
+---------+---------+---------+---------+---------+
```

```
Courses and their preferences, in order from most preferred to least preferred.
Course capacity is denoted by the number in () -- ex: (2) means capacity 2.
+-------------+-------------+-------------+
| CIS 502 (2) | CIS 520 (2) | CIS 521 (2) |
+-------------+-------------+-------------+
| Brianna     | Tommy       | Tommy       |
| Steven      | Sam         | Sam         |
|             | Brianna     | Steven      |
|             | John        | John        |
|             | Steven      |             |
+-------------+-------------+-------------+
```

This results in the following matches:

```
+-------------+-------------+-------------+
| CIS 502 (2) | CIS 520 (2) | CIS 521 (2) |
+-------------+-------------+-------------+
| Brianna     | Tommy       | Steven      |
|             | Sam         | John        |
+-------------+-------------+-------------+
```

Important: For testing purposes, we require your lists to be sorted in order from most preferred, to least preferred. That's why in the list above Tommy is before Sam under CIS 520, and Steven is before John under CIS 521.

If you follow the invariants specified in the stub-files, this ordering should come naturally.

Fun fact: what you're implementing is a simpler version of the algorithm used by the National Resident Matching Program to place U.S. medical school students into residency training programs located in United States hospitals. That algorithm is more complicated though, because among other things it considers married couples who are trying to be matched together. In that case, a stable matching sometimes does not exist.

Implementation Details

`Student.java`

To help you, we've already implemented this class for you. Please read through it to make sure you understand what's available for use. You cannot modify anything in this file as you will not be submitting it.

`Course.java`

Most of this class has been implemented for you. You will have to fill in the two methods indicated by the `TODO: implement` comments, and write test cases.

`ProcessWaitlist.java – makeAssignments`

The algorithm for this method is above, and more comments are in the stub file to help guide you further.

Style & Tests

The above parts together are worth a total of 30 points. Style and code coverage are graded on a subtractive basis with deductions as specified in the *Setup and Logistics* section of the writeup, and you will be graded according to the CIS 121 style guide (/~cis121/current/java_style_guide.html). Gradescope will give you your grade on this section immediately upon submission to either autograder.

IMPORTANT: Please **DO NOT** use any external files (.txt, etc.) to write your JUnit tests for this assignment. Our code coverage tool will fail, and you will not be able to submit your code due to failing test cases.

You will need to write comprehensive unit test cases for each of the classes, inner classes, and methods you implement (including helper methods!) in order to ensure correctness. Make sure you consider edge cases and exceptional cases in addition to typical use cases. Use multiple methods instead of cramming a bunch of asserts into a single test method. Your test cases will be auto-graded for code coverage. Be sure to read the testing guide (/~cis121/current/testing_guide.html) for some pointers on what level of testing we expect from you. Finally, due to the Code Coverage tools we use, **please be sure to use JUnit 4** as it is the only version supported.

Note: You will not be able to write JUnit test cases for any private methods. Instead, make them package-private by leaving off any privacy modifier (i.e., do not write `public` or `private`).

This assignment was developed by Steven Bursztyn in Spring 2019 for CIS 121
Last updated on Thu, Jan 16 at 11:08 PM.