COS 126 Programming Exam 1 (Spring 2021)

Instructions

Download zip now. Download the project zip file, which includes all the files you will need, including this Exam PDF, from Exams tab on the course web page. You should download and open the zip the same way you did for your assignments. Here is a doc with tips on that for both Mac and Windows. These instructions are the PDF in that zip file, and you can also open Intellij on the folder inside the zip (`ZoomRooms`).

Before the exam. You may read this page of instructions before the exam begins. But do not start *(even by reading the next page)* until you are instructed to do so. Also do not open the Java file in IntelliJ until the exam begins.

Zoom. Please remain in zoom during the exam. Leave **video off** and **mute** your mic. If you have a question about the exam content, send a **private** chat to:

- Adam Finkelstein, if your last name begins A-L
- Alan Kaplan, if your last name begins M-Z

Emergencies. If you have an emergency, contact the course admin, Kobi Kaplan, either via Zoom chat or at kskaplan@princeton.edu.

Time. You have 50 minutes to complete the exam and upload a Java file with your final solution.

Recommended plan. This exam follows a step-by-step process. Unlike other exams where you might find it advantageous to read the whole thing before you start, here we recommend you read each step and implement it as you go along. Also you should upload partial solutions a few times during the exam, just as a precaution.

What code to edit. The initial code contains several functions to get you started. We *strongly recommend* that you only add or modify code between the pairs of comments saying "STUDENT CODE BEGIN" and "STUDENT CODE END". This is the easiest way to solve the tasks in this exam. Though it is neither recommended nor needed: you may edit other code and/or add your own functions, provided you **do not** modify the existing function signatures.

Resources. You may use: your textbook, the booksite, your notes, your code from programming assignments and precepts, the code on the COS 126 course website, materials from lectures, labs and precepts, and Ed.

No communication. No form of communication is permitted (e.g., talking, texting, etc.) during the exam, except with course staff.

Do not discuss later. Due to multiple exam times, and various conflicts, some of your peers will take the exam at a different time. Do not discuss the exam contents with anyone *(not even other students that you know already took the exam!)* until after the graded exams are returned.

Honor Code pledge. Before submitting your solution, you must "electronically sign" the honor code in the obvious comment at the end of your Java file, by retyping the pledge and then your name.

Submissions. Submit your work using the `Submit` link on the **Exams** tab of the course web page. You are responsible for submitting your solution before the time is called at the end of the exam.

Grading. Your program will be graded mainly on correctness. You will lose a substantial fraction of your overall grade if your program does not compile, or if it crashes on typical inputs. Clarity (including comments), design, and efficiency are secondary concerns with regards to grading on this timed exam. Nevertheless, writing clear code is always important, and will generally help you understand your own code better.

Background

You are an engineer at a startup company called Zoom, writing software for online video meetings. A feature commonly used in classroom settings is called *breakout rooms*. It allows teachers to divide a large class into smaller groups, each of which meets independently in different "rooms."

Greedy Algorithm. Another engineer has already developed an algorithm to assign students to rooms in a "greedy" fashion, as follows. Suppose the room capacity is *N*. The greedy algorithm places the first *N* students in Room 0, and the next *N* students in Room 1, and so forth, until there are no students left. (Note that as engineers we are numbering the rooms starting with 0 – the convention we use throughout this exam.) Unfortunately there are some drawbacks to this algorithm, and your manager has asked you to provide some alternatives. But first you need to write some helper functions to support this algorithm.

Getting Started

The only file you need to edit is `ZoomRooms.java`, and it contains some code to get you started.

Step 0. Notice that if you compile and run the code it already does something:

```
$ javac-introcs ZoomRooms.java
$ java-introcs ZoomRooms greedy 3

Room: 0
-------
```

There are two command line arguments given: the string `greedy` and the number 3. The first one tells the program which assignment algorithm to use (`greedy` in this case, and the other two options are `robin` and `random`). The second argument is the maximum number of students the teacher wants in each room (3 in this case). You may assume this number is always greater than 1.

Also possibly helpful is a debugging function (called `debug`) that tells you the state of some variables in the main function. It can be triggered by appending `-debug` to the name of the algorithm. So to show this special output, use the algorithm name `greedy-debug`, `robin-debug` or `random-debug`, like this:

```
$ java-introcs ZoomRooms greedy-debug 3
Called debug() function.
roomSize: 3
numRooms: 2
assignedRooms...
0: 0
1: 0
2: 0
3: 1
4: 1
names...
0: Ava
1: Ben
2: Carol
3: Dan
4: Emma
```

You do not need to use this function to write a good solution for this exam, but some people might find it helpful especially if they get stuck. Don't worry too much about the details of that debugging output right now – the different parts will become clear shortly.

Also as a reminder: the code you submit in the end should *still* compile, and also run without errors for all inputs that match the program specications.

The Exam

Step 1. The starting code you ran in Step 0 only prints the first room. Modify the `printRooms` function so that it prints all `numRooms` rooms, by using a loop. If your changes are successful, when you compile and run it, `ZoomRooms` should now print out the headers for Rooms 0 and 1, as shown below. Note that there are still no student names printed yet, which will be addressed in the next step.

```
$ java-introcs ZoomRooms greedy 3

Room: 0
-------

Room: 1
-------
```

Step 2. Next you will modify the `printRoom` function so that it prints names of students in a room, not just the room number. This function has an argument `room` (the room number). It has two other arguments that are *parallel arrays:* `assignedRooms` and `studentNames`. The first array specifies which room each student is assigned to, while the second contains their names. Here's an example of what those arrays might look like in this program:

index	assignedRooms	studentNames
0	0	Ava
1	0	Ben
2	0	Carol
3	1	Dan
4	1	Emma

Write a loop that considers each student in turn, and prints out the names of those students whose entry in `assignedRooms` match the `room` argument. For the example arrays shown above, if `room == 1`, then `printRoom` would output the names Dan and Emma under the dashed line. So running the program again should produce this:

```
$ java-introcs ZoomRooms greedy 3

Room: 0
-------
Ava
Ben
Carol

Room: 1
-------
Dan
Emma
```

Upload through step 2...

This might be a good time to upload your partially completed exam.

Step 3. Your starting code has a function `readNames` returns an array of student names. The temporary code you were given always returns the same five names shown above (Ava, Ben, ...). Now you will change this function so that it reads from `StdIn` an integer *N* ≥ 0, followed by *N* names. You can assume that after *N*, the input will always be exactly *N* names, each on one line that could be read using the function `StdIn.readString`. There are several example data files in the directory that conform to this specification. For example `names3.txt` contains 3 names (Ava, Ben and Carol).

Your function should return an array containing those *N* student names. For example, if you provide the file `names3.txt` on `StdIn` your version of `readNames` would return the three names in that text file, and the output would be:

```
$ java-introcs ZoomRooms greedy 3 < names3.txt

Room: 0
-------
Ava
Ben
Carol

Room: 1
-------
```

Note that with the code we have so far we will always print out exactly two rooms, regardless of the number of students or the room capacity. We will fix that next...

Step 4. Now you have a simple math problem. The second command line argument specifies the room size (the maximum capacity for each room). For example, in the command shown above it's 3 (right after the word "greedy"). Since the room size is 3, it means that Ava, Ben and Carol can all fit in Room 0.

The function `roomsNeeded` should compute the number of rooms needed; but as provided it always just returns 2, no matter what. It takes two arguments: `numStudents` (the total number of students) and `roomSize` (the size of each room, as specified on the command line).

Change this function to return `numStudents` divided by `roomSize`, **rounding up** to an integer in cases where it does not divide evenly. (Hint: there's a useful function the `Math` library.) Now you should get the following output, because only one room is needed:

```
$ java-introcs ZoomRooms greedy 3 < names3.txt

Room: 0
-------
Ava
Ben
Carol
```

To convince yourself that your function works, try it out with a few different inputs. Here are some example outputs that you could verify, and of course you can think of your own:

numRooms	command
0	`java-introcs ZoomRooms greedy-debug 3 < names0.txt`
1	`java-introcs ZoomRooms greedy-debug 3 < names3.txt`
4	`java-introcs ZoomRooms greedy-debug 3 < names11.txt`
9	`java-introcs ZoomRooms greedy-debug 3 < names26.txt`

Upload through Step 4...

This might be a good time to upload your partially completed exam.

Step 5. Now in the function `assignRobin` you will write an algorithm to assign the students in "round robin" order: the first student in Room 0, the second student in Room 1, third in Room 2, and so forth up to the *N*-th student in Room *N-1*. Next it wraps around: the *(N+1)*-th student goes in Room 0, and the *(N+2)*-th in Room 1, etc. After you implement this function, test it with various cases like this one:

```
$ java-introcs ZoomRooms robin 3 < names5.txt

Room: 0
-------
Ava
Carol
Emma

Room: 1
-------
Ben
Dan
```

Just as in previous steps, if you prefer to see the "debug" version you could use something like:

```
$ java-introcs ZoomRooms robin-debug 3 < names5.txt
```

Upload through Step 5...

This might be a good time to upload your largely completed exam.

Bonus Challenge

Step 6. Congratulations if you made it this far! This last step is difficult on a timed exam. It is meant to provide an extra challenge only for those of you who have sufficient time for it. You will only receive credit for this step if your previous steps are all solved correctly; and even then a perfect solution to this step will only be worth 5% of the overall exam grade. So only attempt this step if you are confident in your previous solutions.

Your goal here is to implement a third assignment algorithm in the function `assignRandom` that places students in rooms randomly, with equal probability of ending up in any room. Note that rooms still have capacity `roomSize`, so you need to keep track of that somehow. As a hint, here are two possible general strategies for accomplishing this goal:

- As you assign students to rooms (randomly) you could maintain an array that tracks how many are in each room, and avoid assigning students to rooms that are full.
- Alternately, you could shuffle the list of students randomly; and then assign them using, say, the "round robin" approach.

Either way would work, and you might also think of other strategies. Regardless, the details are up to you. Obviously if you implement this algorithm it should generally give different answers each time you run it on the same input. For example, try this a few times and check the outputs:

```
$ java-introcs ZoomRooms random 3 < names5.txt
```

Finishing Up

Don't forget to write and electronically sign the Honor Code pledge in the comment at the end of your Java file, before uploading the final version.

After the Exam

This is optional enrichment, intended for you to think about after the exam is completed.

What are the relative benefits of the three algorithms described? One factor to consider: in what scenarios could each algorithm leave an empty room; or one student alone in a room? Is one algorithm more or less efficient for handling large groups of students? What are the other tradeoffs?

New debugging strategy. Notice in this exam we used a new debugging strategy that you may not have seen before. We made a function called `debug` that printed out the most important data from the program. You might like to use this idea to help you in future programming assignments.

Compiled on Thu Mar 11 07:52:31 EST 2021