RU Kindergarten – 100 course points

The purpose of this assignment is to practice your understanding of linked list structures.

This assignment will take longer to complete than the first assignment.

Start your assignment early! You need time to understand the assignment and to answer the many questions that will arise as you read the description and the code provided.

Refer to our Programming Assignments FAQ for instructions on how to install VSCode, how to use the command line and how to submit your assignments.

Overview

In this assignment you will simulate activities in a kindergarten classroom.

You will simulate the students on a line, the students on their seats, and the students playing musical chairs.

Chinese New Year

Implementation

Overview of files provided

- **Student** class which holds a student's information.
- **SNode** class represents the node object to be used in the linked structures. It contains a reference to a Student object and a reference to the next node in the list.
- **Classroom class** holds all of the methods to be written and that will be tested when running the game. Edit the empty methods with you solution, but **DO NOT edit the provided ones or the methods signatures of any method**. This is the file you submit.
- **Driver** class, which is used to test your methods interactively. The classroom state will be printed after every method is used. Feel free to edit this file, as it is provided only to help you test your code. It is not submitted and it is not used to grade your code.
- **StdRandom** class contains the *StdRandom.uniform(n)* method that you will use in the method that plays the musical chairs game.
- **StdIn** and **StdOut**, which are used by the driver. **Do not edit these classes.**.
- Multiple text files. Feel free to edit them or even make new ones to help test your code. They are not submitted.
    - Files containing student information.
    - Files containing seat availability in the classroom.

Classroom.java

- DO NOT add new import statements.
- DO NOT change any of the method's signatures.
- You are encouraged to write helper methods as you see fit. Just make sure that they are private.

The class contains:

- **studentInLine** which refers to the first student in the singly linked list
- **musicalChairs** which refers to the LAST student in the circularly linked list when the game is being played
- **seatingAvailability** which is a 2D boolean array that shows which seats are available for students to seat at
- **studentsSitting** which is a 2D Student array that contains the students when they are sitting
- NOTE: seatingAvailability and studentsSitting are parallel arrays meaning that seatingAvailability[i][j] also refers to the same seat in studentsSitting[i][j]
- provided methods that are used by the driver and empty methods you are expected to write your code in

- the printClassroom() function can be used to show the state of the classroom (eg. the states of students in line, seating, and musical chairs) at any time in your code

Methods to be implemented by you:

1. makeClassroom

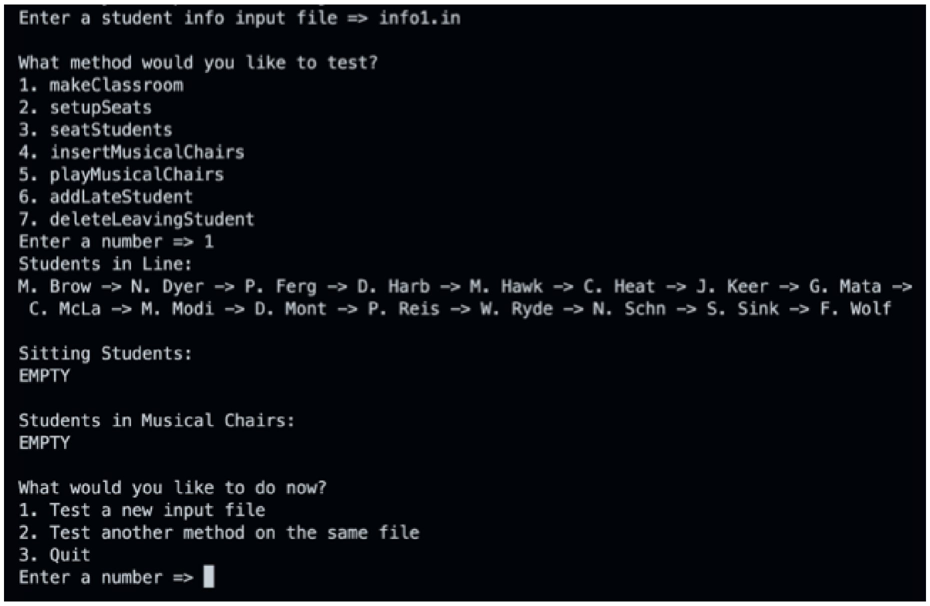
This method simulates students coming into the classroom and standing in line. You have been provided some input files to test this method (info1.in, info2.in, info3.in, info4.in). The format is as follows:

- One line containing an integer representing the number of students in the file (one per line)
- Several lines (one for student) containing students first name, last name, and height, space separated

Use the **StdIn** library to read from a file:

- `StdIn.setFile(filename)` opens a file to be read
- `StdIn.readInt()` reads the next integer value from the opened file (weather the value is in the current line or in the next line)
- `StdIn.readString()` read the next String value from the opened file

This method creates students listed on the input file and inserts students in ALPHABETICAL ORDER into the singularly linked list **studentsInLine**. ~~Students with the same last name will be inserted in order of insertion~~. Use the *compareNameTo* method provided in the Student class to compare two Students name alphabetically. This is the expected output for info1.in



```
Enter a student info input file => info1.in

What method would you like to test?
1. makeClassroom
2. setupSeats
3. seatStudents
4. insertMusicalChairs
5. playMusicalChairs
6. addLateStudent
7. deleteLeavingStudent
Enter a number => 1
Students in Line:
M. Brow -> N. Dyer -> P. Ferg -> D. Harb -> M. Hawk -> C. Heat -> J. Keer -> G. Mata ->
 C. McLa -> M. Modi -> D. Mont -> P. Reis -> W. Ryde -> N. Schn -> S. Sink -> F. Wolf

Sitting Students:
EMPTY

Students in Musical Chairs:
EMPTY

What would you like to do now?
1. Test a new input file
2. Test another method on the same file
3. Quit
Enter a number =>
```

2. setupSeats

This method creates the classroom seating availability.

Reads the seating availability from the input file (seating1.in, seating2.in). The input file format is as follows:

- The first line contains an integer representing the number of rows in the classroom, say r
- The second line contains an integer representing the number of columns in the classroom, say c
- Number of r lines, each containing c true or false values (true denotes an available seat, false denotes that there is no seat at that position in the classroom)

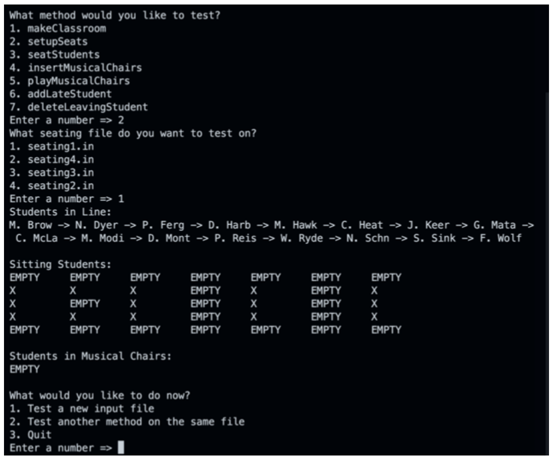
Use the **StdIn** library to read from a file:

- `StdIn.setFile(filename)` opens a file to be read
- `StdIn.readInt()` reads the next integer value from the opened file (weather the value is in the current line or in the next line)
- `StdIn.readBoolean()` read the next boolean value from the opened file (weather the value is in the current line or in the next line)

This method creates and populates the **seatingAvailability** array, it also creates the **studentsSitting** array with the same number of rows and columns as the seatingAvailability array

This method does not seat students on the seats.

This is the expected output for info1.in and seating1.in:



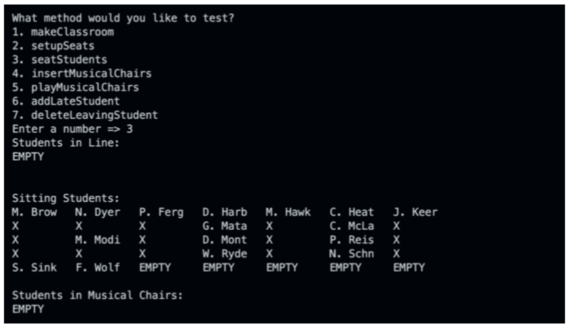
```
What method would you like to test?
1. makeClassroom
2. setupSeats
3. seatStudents
4. insertMusicalChairs
5. playMusicalChairs
6. addLateStudent
7. deleteLeavingStudent
Enter a number => 2
What seating file do you want to test on?
1. seating1.in
2. seating4.in
3. seating3.in
4. seating2.in
Enter a number => 1
Students in Line:
M. Brow -> N. Dyer -> P. Ferg -> D. Harb -> M. Hawk -> C. Heat -> J. Keer -> G. Mata ->
 C. McLa -> M. Modi -> D. Mont -> P. Reis -> W. Ryde -> N. Schn -> S. Sink -> F. Wolf

Sitting Students:
EMPTY     EMPTY     EMPTY     EMPTY     EMPTY     EMPTY     EMPTY
X         X         X         EMPTY     X         EMPTY     X
X         EMPTY     X         EMPTY     X         EMPTY     X
X         X         X         EMPTY     X         EMPTY     X
EMPTY     EMPTY     EMPTY     EMPTY     EMPTY     EMPTY     EMPTY

Students in Musical Chairs:
EMPTY

What would you like to do now?
1. Test a new input file
2. Test another method on the same file
3. Quit
Enter a number => 
```

3. seatStudents

This method simulates students taking their seats in the Kindergarten classroom. Assume that the students are currently in **studentsInLine** and that there are enough available seats to seat all student in line.

- Students will be seated starting at the first row, first seat (studentsSitting[0][0] if available). Once the first row is filled, continues into the next row.
- It uses the seatingAvailability array to determine if a seat is open. A student can sit at seat studentsSitting[i][j] only if seatingAvailability[i][j] is true (seat available).
- First seat any remaining students from the **musicalChairs**. There will be at most one student, the game's winner.
- Then seat the first student in **studentsInLine** and moves on to the second, third and so on.

NOTE: a student is only in one place (musical chairs, seating chairs, or in line) at a time.



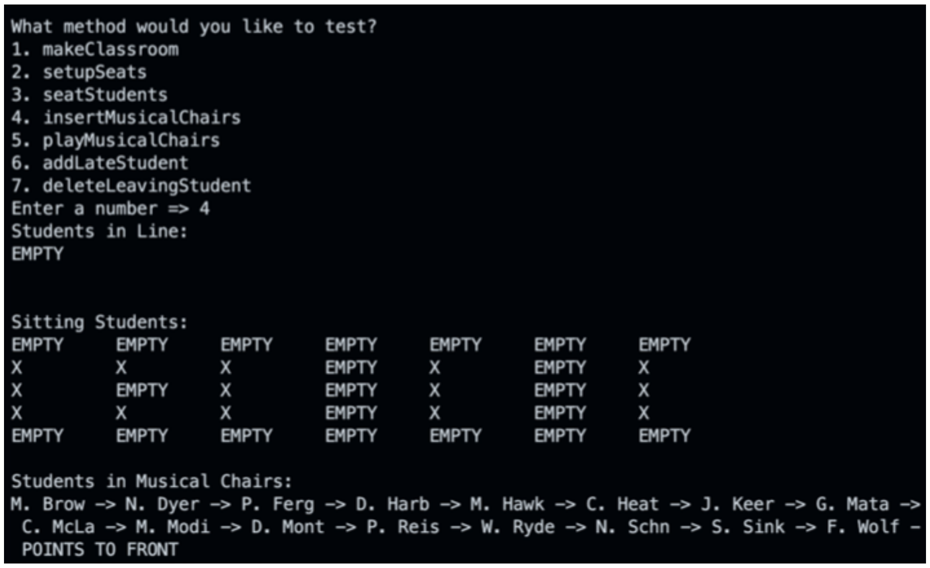
```
What method would you like to test?
1. makeClassroom
2. setupSeats
3. seatStudents
4. insertMusicalChairs
5. playMusicalChairs
6. addLateStudent
7. deleteLeavingStudent
Enter a number => 3
Students in Line:
EMPTY


Sitting Students:
M. Brow   N. Dyer   P. Ferg   D. Harb   M. Hawk   C. Heat   J. Keer
X         X         X         G. Mata   X         C. McLa   X
X         M. Modi   X         D. Mont   X         P. Reis   X
X         X         X         W. Ryde   X         N. Schn   X
S. Sink   F. Wolf   EMPTY     EMPTY     EMPTY     EMPTY     EMPTY

Students in Musical Chairs:
EMPTY
```

4. insertMusicalChairs

This method represents students preparing to start musical chairs! Assume that the students are in **studentsSitting**.

- Imagine this as a circle of chairs.
- This method will take students from the **studentsSitting** array and add them to the **musicalChairs** circular linked list.
- Students are to be inserted at the end of the linked list by traversing row-wise, then column-wise in the studentsSitting array.
- REMEMBER: the pointer to a circular linked list points to the LAST item in the list, and that item points to the front.

This is the expected output using info1.in and seating1.in:

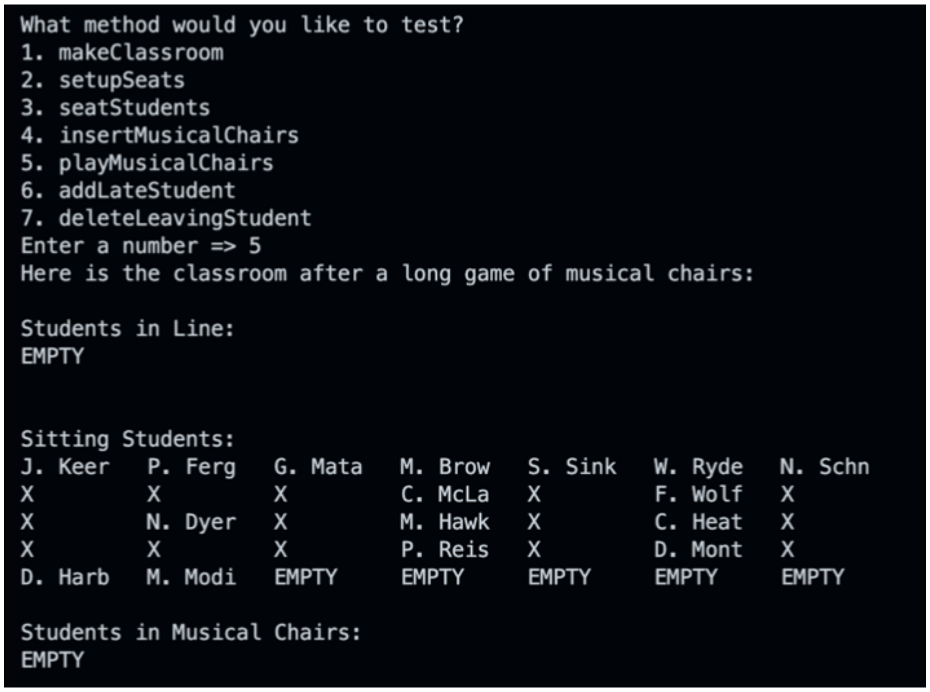


```
What method would you like to test?
1. makeClassroom
2. setupSeats
3. seatStudents
4. insertMusicalChairs
5. playMusicalChairs
6. addLateStudent
7. deleteLeavingStudent
Enter a number => 4
Students in Line:
EMPTY


Sitting Students:
EMPTY    EMPTY    EMPTY    EMPTY    EMPTY    EMPTY    EMPTY
X        X        X        EMPTY    X        EMPTY    X
X        EMPTY    X        EMPTY    X        EMPTY    X
X        X        X        EMPTY    X        EMPTY    X
EMPTY    EMPTY    EMPTY    EMPTY    EMPTY    EMPTY    EMPTY

Students in Musical Chairs:
M. Brow -> N. Dyer -> P. Ferg -> D. Harb -> M. Hawk -> C. Heat -> J. Keer -> G. Mata ->
 C. McLa -> M. Modi -> D. Mont -> P. Reis -> W. Ryde -> N. Schn -> S. Sink -> F. Wolf -
 POINTS TO FRONT
```

5. playMusicalChairs

This method simulates students playing musical chairs! Assume the students are in **musicalChairs**.

- This method “eliminates” a student from the game until a final player is left
- Each player that is “eliminated” is then placed back in studentsInLine
- **Complete insertMusicalChairs before starting this method**
- Use *StdRandom.uniform(x)*, where x is the number of students in line, to get a number ***n*** between 0 (inclusive) and *x (*exclusive). The student at position ***n*** is to be eliminated from musicalChairs.
    - Notice that the driver is setting a seed for the random number generator. The seed value is 2022.
- Eliminated students are placed in studentsInLine IN HEIGHT ORDER (shortest to tallest). If students have the same height, add them in order of insertion, after.
- One student will remain in musicalChairs, this is the winner! **The winner doesn’t leave the musical chairs to enter the line.**
- Finally, seatStudents() should be called to place students in their seats.
    - the winner IS TO BE seated in the first available seat,
    - then students from **studentsInLine** will be seated in height order, because the students in this linked list ARE ALREADY in ascending height order (see playMusicalChairs method).



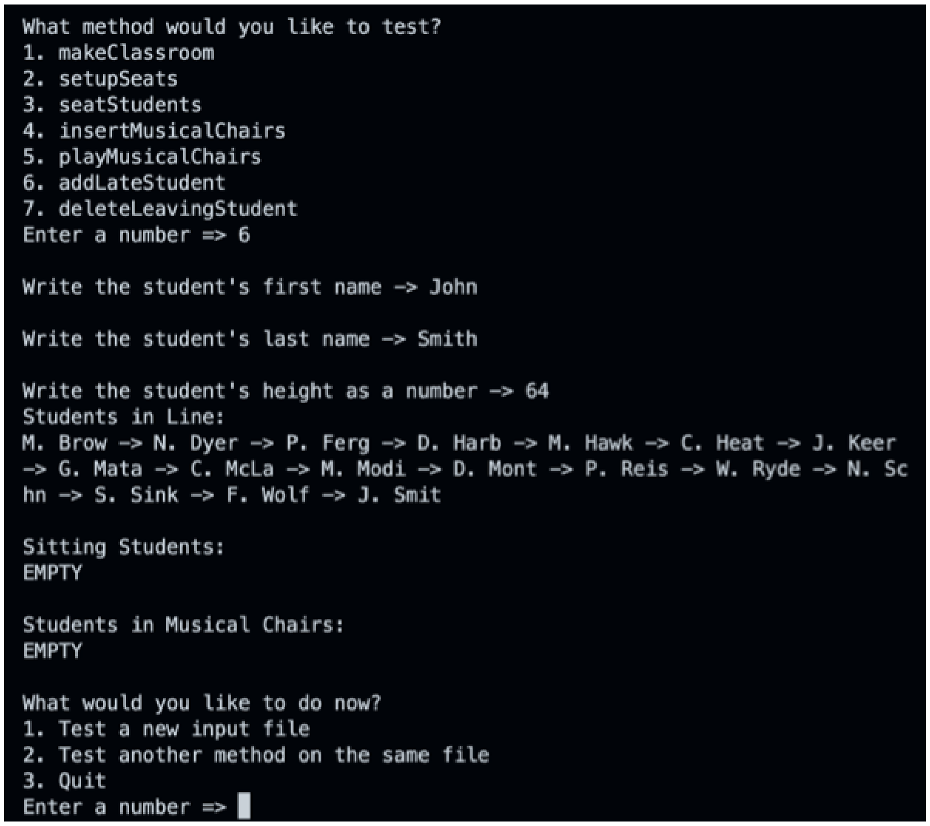
```
What method would you like to test?
1. makeClassroom
2. setupSeats
3. seatStudents
4. insertMusicalChairs
5. playMusicalChairs
6. addLateStudent
7. deleteLeavingStudent
Enter a number => 5
Here is the classroom after a long game of musical chairs:

Students in Line:
EMPTY


Sitting Students:
J. Keer   P. Ferg   G. Mata   M. Brow   S. Sink   W. Ryde   N. Schn
X         X         X         C. McLa   X         F. Wolf   X
X         N. Dyer   X         M. Hawk   X         C. Heat   X
X         X         X         P. Reis   X         D. Mont   X
D. Harb   M. Modi   EMPTY     EMPTY     EMPTY     EMPTY     EMPTY

Students in Musical Chairs:
EMPTY
```

6. addLateStudent

This method simulates a student arriving late to school. Assume the students could be anywhere in the classroom.

- This method's input is a string of the student's first name, a string of the student's last name, and an integer of the student's height.
- Wherever the students are, this method will add this student to the group:
    - If students are in line or at musical chairs, insert this student as the last node of the linked list
    - If students are sitting, insert this student at the first available seat in the **studentsSitting** 2D array
- Assume the late student is not the first to arrive.

Here is an example output using info1.in and running makeClassroom and addLateStudent for John Smith with height 64:



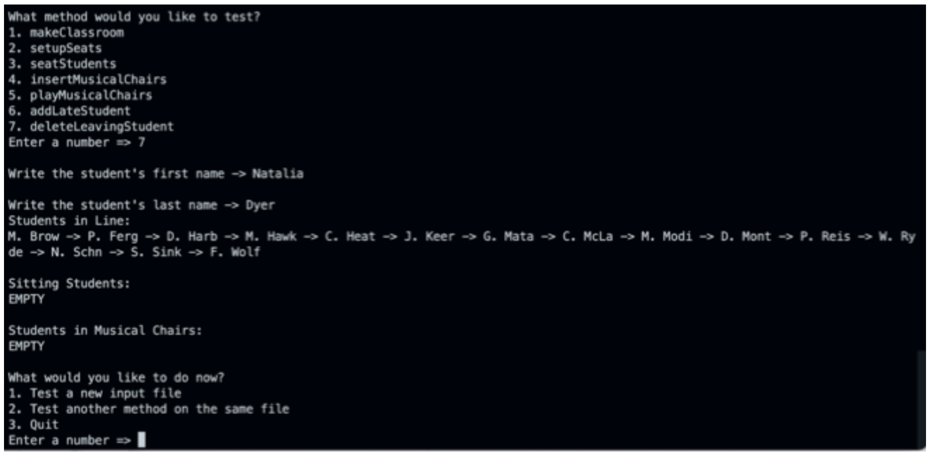
```
What method would you like to test?
1. makeClassroom
2. setupSeats
3. seatStudents
4. insertMusicalChairs
5. playMusicalChairs
6. addLateStudent
7. deleteLeavingStudent
Enter a number => 6

Write the student's first name -> John

Write the student's last name -> Smith

Write the student's height as a number -> 64
Students in Line:
M. Brow -> N. Dyer -> P. Ferg -> D. Harb -> M. Hawk -> C. Heat -> J. Keer
-> G. Mata -> C. McLa -> M. Modi -> D. Mont -> P. Reis -> W. Ryde -> N. Sc
hn -> S. Sink -> F. Wolf -> J. Smit

Sitting Students:
EMPTY

Students in Musical Chairs:
EMPTY

What would you like to do now?
1. Test a new input file
2. Test another method on the same file
3. Quit
Enter a number =>
```

7. deleteLeavingStudent

This method simulates a student leaving school early. Assume the students could be anywhere in the classroom.

- This method’s input is a string of the student’s first name and a string of the student’s last name. **This is case-INsensitive!**
- Wherever the students are, this method will delete this student from the group.
  - If students are in line or at musical chairs, delete this student’s node from that linked list.
  - If students are sitting, make this student’s seat in **studentsSitting** null.

Here is an example of deleting “Natalia Dyer” from studentsInLine in info1.in.



```
What method would you like to test?
1. makeClassroom
2. setupSeats
3. seatStudents
4. insertMusicalChairs
5. playMusicalChairs
6. addLateStudent
7. deleteLeavingStudent
Enter a number => 7

Write the student's first name -> Natalia

Write the student's last name -> Dyer
Students in Line:
M. Brow -> P. Ferg -> D. Harb -> M. Hawk -> C. Heat -> J. Keer -> G. Mata -> C. McLa -> M. Modi -> D. Mont -> P. Reis -> W. Ry
de -> N. Schn -> S. Sink -> F. Wolf

Sitting Students:
EMPTY

Students in Musical Chairs:
EMPTY

What would you like to do now?
1. Test a new input file
2. Test another method on the same file
3. Quit
Enter a number => 
```

Implementation Notes

- YOU MAY only update the methods with the WRITE YOUR CODE HERE line.
- DO NOT add any instance variables to the Classroom class.
- DO NOT add any public methods to the Classroom class.
- YOU MAY add private methods to the Classroom class.
- **DO NOT** use System.exit()

VSCode Extensions

You can install VSCode extension packs for Java. Take a look at this tutorial. We suggest:

- Extension Pack for Java
- Project Manager for Java
- Debugger for Java

Importing VSCode Project

1. Download RUKindergarten.zip from Autolab Attachments.
2. Unzip the file by double clicking.
3. Open VSCode
   - Import the folder to a workspace through **File** > **Open**

Executing and Debugging

- You can run your program through VSCode or you can use the Terminal to compile and execute. We suggest running through VSCode because it will give you the option to debug.
- How to debug your code
- If you choose the Terminal:
  - first navigate to **RUKindergarten** directory/folder

    - to compile: **javac -d bin src/kindergarten/*.java**
    - to execute: **java -cp bin kindergarten.Driver**

Before submission

Collaboration policy. Read our collaboration policy here.

Submitting the assignment. Submit *Classroom.java* separately via the web submission system called Autolab. To do this, click the *Assignments* link from the course website; click the *Submit* link for that assignment.

Getting help

If anything is unclear, don't hesitate to drop by office hours or post a question on Piazza.

- Find instructors and head TAs office hours *here*
- Find tutors office hours on Canvas -> Tutoring -> RU CATS
- In addition to office hours we have the CAVE (Collaborative Academic Versatile Environment), a community space staffed with lab assistants which are undergraduate students further along the CS major to answer questions.

Problem by Ethan Chou, Maksims Kurjanovics Kravcenko, and Kal Pandit