FRQ Type 4 - Seating Chart

1. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

   Notes:

   - Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
   - In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

   A student in a school is represented by the following class.

```
public class Student
{
  /** Returns the name of this Student. */
  public String getName()
  {  /* implementation not shown */  }

  /** Returns the number of times this Student has missed class. */
  public int getAbsenceCount()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

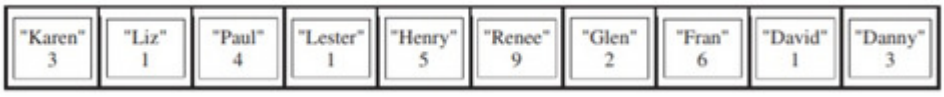
   The class SeatingChart, shown below, uses a two-dimensional array to represent the seating arrangement of students in a classroom. The seats in the classroom are in a rectangular arrangement of rows and columns.

FRQ Type 4 - Seating Chart

```
public class SeatingChart
{
  /** seats[r][c] represents the Student in row r and column c in the classroom. */
  private Student[][] seats;

  /** Creates a seating chart with the given number of rows and columns from the students in
   *  studentList. Empty seats in the seating chart are represented by null.
   *  @param rows the number of rows of seats in the classroom
   *  @param cols the number of columns of seats in the classroom
   *  Precondition: rows > 0; cols > 0;
   *                rows * cols >= studentList.size()
   *  Postcondition:
   *    - Students appear in the seating chart in the same order as they appear
   *      in studentList, starting at seats[0][0].
   *    - seats is filled column by column from studentList, followed by any
   *      empty seats (represented by null).
   *    - studentList is unchanged.
   */
  public SeatingChart(List<Student> studentList,
                      int rows, int cols)
  {  /* to be implemented in part (a) */  }

  /** Removes students who have more than a given number of absences from the
   *  seating chart, replacing those entries in the seating chart with null
   *  and returns the number of students removed.
   *  @param allowedAbsences an integer >= 0
   *  @return number of students removed from seats
   *  Postcondition:
   *    - All students with allowedAbsences or fewer are in their original positions in seats.
   *    - No student in seats has more than allowedAbsences absences.
   *    - Entries without students contain null.
   */
  public int removeAbsentStudents(int allowedAbsences)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

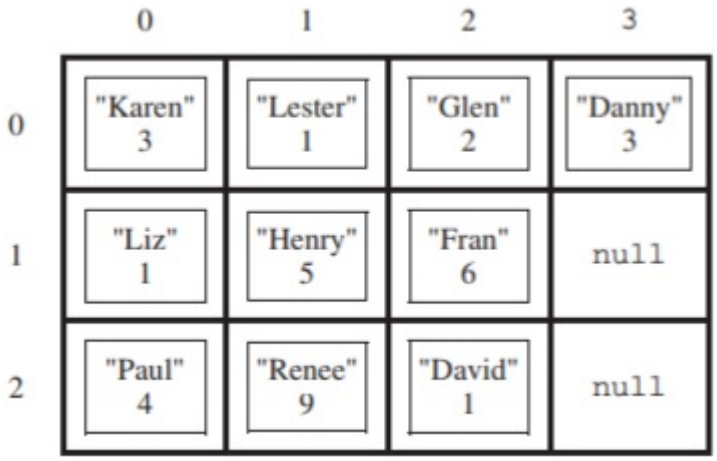
a. Write the constructor for the SeatingChart class. The constructor initializes the seats instance variable to a two-dimensional array with the given number of rows and columns. The students in studentList are copied into the seating chart in the order in which they appear in studentList. The students are assigned to consecutive locations in the array seats, starting at seats[0][0] and filling the array column by column. Empty seats in the seating chart are represented by null.

For example, suppose a variable List roster contains references to Student objects in the following order.



FRQ Type 4 - Seating Chart

A SeatingChart object created with the call new SeatingChart(roster, 3, 4) would have seats initialized with the following values.

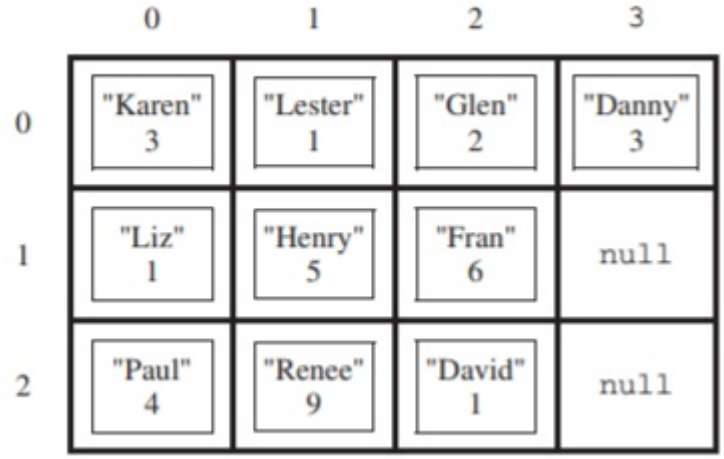


Complete the SeatingChart constructor below.

```
/** Creates a seating chart with the given number of rows and columns from the students in
 *  studentList. Empty seats in the seating chart are represented by null.
 *  @param rows the number of rows of seats in the classroom
 *  @param cols the number of columns of seats in the classroom
 *  Precondition: rows > 0; cols > 0;
 *                rows * cols >= studentList.size()
 *  Postcondition:
 *    - Students appear in the seating chart in the same order as they appear
 *      in studentList, starting at seats[0][0].
 *    - seats is filled column by column from studentList, followed by any
 *      empty seats (represented by null).
 *    - studentList is unchanged.
 */
public SeatingChart(List<Student> studentList,
                    int rows, int cols)
```

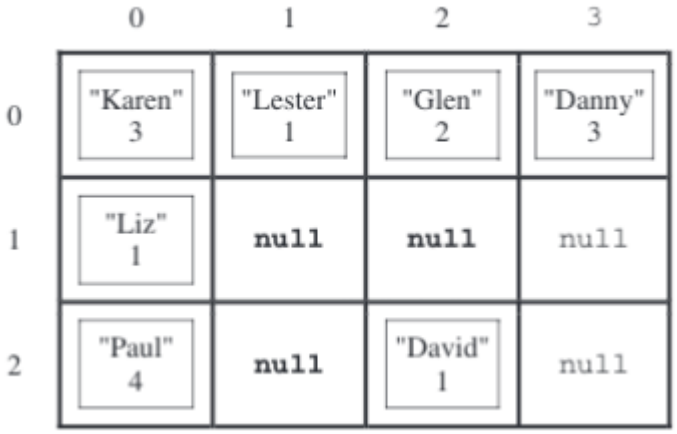
b. Write the removeAbsentStudents method, which removes students who have more than a given number of absences from the seating chart and returns the number of students that were removed. When a student is removed from the seating chart, a null is placed in the entry for that student in the array seats. For example, suppose the variable SeatingChart introCS has been created such that the array seats contains the following entries showing both students and their number of absences.

FRQ Type 4 - Seating Chart



	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	"Henry" 5	"Fran" 6	null
2	"Paul" 4	"Renee" 9	"David" 1	null

After the call introCS.removeAbsentStudents(4) has executed, the array seats would contain the following values and the method would return the value 3.



	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	**null**	**null**	null
2	"Paul" 4	**null**	"David" 1	null

Class information repeated from the beginning of the question:

```
public class Student

public String getName()
public int getAbsenceCount()

public class SeatingChart

private Student[][] seats
public SeatingChart(List<Student> studentList,
                    int rows, int cols)
public int removeAbsentStudents(int allowedAbsences)
```

Complete method removeAbsentStudents below.

FRQ Type 4 - Seating Chart

```
/** Removes students who have more than a given number of absences from the
 *  seating chart, replacing those entries in the seating chart with null
 *  and returns the number of students removed.
 *  @param allowedAbsences an integer >= 0
 *  @return number of students removed from seats
 *  Postcondition:
 *    - All students with allowedAbsences or fewer are in their original positions in seats.
 *    - No student in seats has more than allowedAbsences absences.
 *    - Entries without students contain null.
 */
public int removeAbsentStudents(int allowedAbsences)
```

Part A: SeatingChart constructor

5 points:

Intent: *Create* SeatingChart *object from list of students*

- **+1** seats = new Student[rows][cols]; (or equivalent code)
- **+1** Accesses all elements of studentList (*no bounds errors on studentList*)
- **+1** Accesses all necessary elements of seats array (*no bounds errors on seats array, point lost if access not column-major order*)
- **+1** Assigns value from studentList to at least one element in seats array
- **+1** On exit: All elements of seats have correct values (*minor loop bounds errors ok*)

Question-Specific Penalties (only deduct once per question)**:**

- **–2** (v) Consistently uses incorrect array name instead of seats or studentList

✓

0	1	2	3	4	5

The student response earns five of the following points:

5 points:

Intent: *Create* SeatingChart *object from list of students*

- **+1** seats = new Student[rows][cols]; (or equivalent code)
- **+1** Accesses all elements of studentList (*no bounds errors on studentList*)
- **+1** Accesses all necessary elements of seats array (*no bounds errors on seats array, point lost if access not column-major order*)

FRQ Type 4 - Seating Chart

- **+1** Assigns value from studentList to at least one element in seats array
- **+1** On exit: All elements of seats have correct values (*minor loop bounds errors ok*)

Question-Specific Penalties (only deduct once per question)**:**

- **–2** (v) Consistently uses incorrect array name instead of seats or studentList

Part B: removeAbsentStudents

4 points:

Intent: *Remove students with more than given number of absences from seating chart and return count of students removed*

- **+1** Accesses all elements of seats (*no bounds errors*)
- **+1** Calls getAbsenceCount() on Student object (*point lost if null case not handled correctly*)
- **+1** Assigns null to all elements in seats array when absence count for occupying student > allowedAbsences (*point lost if seats array element changed in other cases*)
- **+1** Computes and returns correct number of students removed

Question-Specific Penalties (only deduct once per question)**:**

- **–2** (v) Consistently uses incorrect array name instead of seats or studentList

0	1	2	3	4

The student response earns four of the following points:

4 points:

Intent: *Remove students with more than given number of absences from seating chart and return count of students removed*

- **+1** Accesses all elements of seats (*no bounds errors*)
- **+1** Calls getAbsenceCount() on Student object (*point lost if null case not handled correctly*)
- **+1** Assigns null to all elements in seats array when absence count for occupying student > allowedAbsences (*point lost if seats array element changed in other cases*)
- **+1** Computes and returns correct number of students removed

Question-Specific Penalties (only deduct once per question)**:**

- **–2** (v) Consistently uses incorrect array name instead of seats or studentList