3. 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.

```
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<Student> roster` contains references to `Student` objects in the following order.

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

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

	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`

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Part (a) continues on page 9.

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)
```

Part (b) begins on page 10.

(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.

	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.

```
/** 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)
```