

2014 AP[®] COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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.
}
```

2014 AP[®] COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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.

AP[®] COMPUTER SCIENCE A

2014 SCORING GUIDELINES

Question 3: Seating Chart

Part (a)	<code>SeatingChart</code> 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*)

Part (b)	<code>removeAbsentStudents</code>	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

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

AP[®] COMPUTER SCIENCE A

2014 CANONICAL SOLUTIONS

Question 3: SeatingChart

Part (a):

```
public SeatingChart(List<Student> studentList, int rows, int cols){
    seats=new Student[rows][cols];
    int studentIndex=0;
    for (int col = 0; col < cols; col++){
        for (int row = 0; row < rows; row++){
            if (studentIndex < studentList.size()){
                seats[row][col] = studentList.get(studentIndex);
                studentIndex++;
            }
        }
    }
}
```

Part (a) alternate:

```
public SeatingChart(List<Student> studentList, int rows, int cols){
    seats=new Student[rows][cols];
    int row=0;
    int col=0;
    for (Student student : studentList){
        seats[row][col]=student;
        row++;
        if (row==rows){
            row=0;
            col++;
        }
    }
}
```

Part (b):

```
public int removeAbsentStudents(int allowedAbsences){
    int count = 0;
    for (int row=0; row < seats.length; row++){
        for (int col=0; col < seats[0].length; col++){
            if (seats[row][col] != null &&
                seats[row][col].getAbsenceCount() > allowedAbsences){
                seats[row][col]=null;
                count++;
            }
        }
    }
    return count;
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.