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

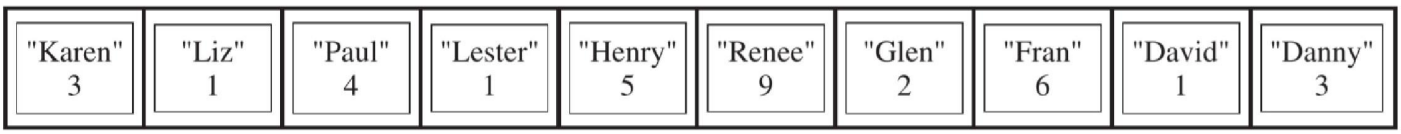
```
public class Student
{
  /** Returns the name of this Student. */
  public String getName()
  {  /* implementation not shown */  }

  /** 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.length
   *  Postcondition:
   *     - Students appear in the seating chart in random order
   *
   *     - seats is filled column by column from studentList, followed by any
   *       empty seats (represented by null).
   *     - studentList is unchanged.
   */
  public SeatingChart(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 random order.The length of `studentList` is smaller or equals to the space numbers in the array `seats`.
Empty seats in the seating chart are represented by `null`.

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




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

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

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.length
 *  Postcondition:
 *     - Students appear in the seating chart in the random order
 *
 *     - seats is filled randmly from studentList, followed by any
 *       empty seats (represented by null).
 *     - studentList is unchanged.
 */
public SeatingChart(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.

| | 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(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)
```

2. Consider a guessing game in which a player tries to guess a hidden word. The hidden word contains only capital letters or numbers and has a length known to the player. A guess contains only capital letters or numbers and has the same length as the hidden word.Neither the constructor nor any of the methods should alter the parameter passed to the constructor, but your implementation may copy the contents of the array.
After a guess is made, the player is given a hint that is based on a comparison between the hidden word and the guess. Each position in the hint contains a character that corresponds to the letter in the same position in the guess. The following rules determine the characters that appear in the hint.

| **If the letter in the guess is …** | **the corresponding character in the hint is** |
|---|---|
| also in the same position in the hidden word, | the matching letter |
| also in the hidden word, but in a different position, | `"+"` |
| not in the hidden word, | `""` |

The `HiddenWord` class will be used to represent the hidden word in the game. The hidden word is passed to the constructor. The class contains a method, `getHint`, that takes a guess and produces a hint.

For example, suppose the variable `puzzle` is declared as follows.

```
HiddenWord puzzle = new HiddenWord(new ArrayList<String>{"H","A","13","P","5"});
```

The following table shows several guesses and the hints that would be produced.

| **Call to `getHint`** | **String returned** |
|---|---|
| `puzzle.getHint(new ArrayList<String>{"A","A","A","A","A"})` | `+A+++` |
| `puzzle.getHint(new ArrayList<String>{"H","E","L","3","O"})` | `H""""` |
| `puzzle.getHint(new ArrayList<String>{"H","E","A","13","T"})` | `H"++"` |
| `puzzle.getHint(new ArrayList<String>{"H","A","13","M5","54"})` | `HA13""` |
| `puzzle.getHint(new ArrayList<String>{"H","A","13","P","5")` | `HA13P5` |

Write the complete `HiddenWord` class, including any necessary instance variables, its constructor, and the method, `getHint`, described above. You may assume that the length of the guess is the same as the length of the hidden word.