ArrayList Practice

QUESTION 1:

Consider a system for processing student test scores. The following class will be used as part of this system and contains a student's name and the student's answers for a multiple-choice test. The answers are represented as strings of length one with an omitted answer being represented by a string containing a single question mark ("?"). These answers are stored in an ArrayList in which the position of the answer corresponds to the question number on the test (question numbers start at 0). A student's score on the test is computed by comparing the student's answers with the corresponding answers in the answer key for the test. One point is awarded for each correct answer and 1⁄4 of a point is deducted for each incorrect answer. Omitted answers (indicated by"?") do not change the student's score.

```
public class StudentAnswerSheet
{
   private ArrayList<String> answers;   // the list of the student's answers


   /** @param key  the list of correct answers, represented as strings of length one
    *            Precondition:  key.size()  is equal to the number of answers in this answer sheet
    *   @return  this student's test score
    */
   public double getScore(ArrayList<String> key)
   {   /*  to be implemented in part (a)  */   }


   /** @return  the name of the student
    */
   public String getName()
   {   /*  implementation not shown  */   }

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

The following table shows an example of an answer key, a student's answers, and the corresponding point values that would be awarded for the student's answers. In this example, there are six correct answers, three incorrect answers, and one omitted answer. The student's score is ((6 * 1) - (3 * 0.25)) = 5.25 .

Question number	0	1	2	3	4	5	6	7	8	9
key	"A"	"C"	"D"	"E"	"B"	"C"	"E"	"B"	"B"	"C"
answers	"A"	"B"	"D"	"E"	"A"	"C"	"?"	"B"	"D"	"C"
Points awarded	1	–0.25	1	1	–0.25	1	0	1	–0.25	1

Consider the following class that represents the test results of a group of students that took a multiple-choice test.

```
public class TestResults
{
  private ArrayList<StudentAnswerSheet> sheets;

  /** Precondition: sheets.size() > 0;
   *               all answer sheets in sheets have the same number of answers
   *  @param key the list of correct answers represented as strings of length one
   *             Precondition: key.size() is equal to the number of answers
   *                           in each of the answer sheets in sheets
   *  @return the name of the student with the highest score
   */
  public String highestScoringStudent(ArrayList<String> key)
  { /* to be implemented in part (b) */ }

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

B) Write the TestResults method highestScoringStudent, which returns the name of the student who received the highest score on the test represented by the parameter key. If there is more than one student with the highest score, the name of any one of these highest-scoring students may be returned. You may assume that the size of each answer sheet represented in the ArrayList sheets is equal to the size of the ArrayList key.

In writing highestScoringStudent, assume that getScore works as specified, regardless of what you wrote in part (a).

Complete method highestScoringStudent below.

```
/** Precondition: sheets.size() > 0;
 *               all answer sheets in sheets have the same number of answers
 *  @param key the list of correct answers represented as strings of length one
 *             Precondition: key.size() is equal to the number of answers
 *                           in each of the answer sheets in sheets
 *  @return the name of the student with the highest score
 */
public String highestScoringStudent(ArrayList<String> key)
```

QUESTION 2:

. Consider a guessing game in which a player tries to guess a hidden word. The hidden word contains only capital letters and has a length known to the player. A guess contains only capital letters and has the same length as the hidden word.

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("HARPS");
```

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

Call to `getHint`	**String returned**
`puzzle.getHint("AAAAA")`	`"+A+++"`
`puzzle.getHint("HELLO")`	`"H****"`
`puzzle.getHint("HEART")`	`"H*++*"`
`puzzle.getHint("HARMS")`	`"HAR*S"`
`puzzle.getHint("HARPS")`	`"HARPS"`

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.