3. 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 ¼ 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

(a) Write the `StudentAnswerSheet` method `getScore`. The parameter passed to method `getScore` is an `ArrayList` of strings representing the correct answer key for the test being scored. The method computes and returns a `double` that represents the score for the student's test answers when compared with the answer key. One point is awarded for each correct answer and ¼ of a point is deducted for each incorrect answer. Omitted answers (indicated by `"?"`) do not change the student's score.

Complete method `getScore` below.

```
/** @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)
```

(b) 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.
}
```

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)
```

4. In this question, you will complete methods in classes that can be used to represent a multi-player game. You will be able to implement these methods without knowing the specific game or the players' strategies.

   The `GameState` interface describes the current state of the game. Different implementations of the interface can be used to play different games. For example, the state of a checkers game would include the positions of all the pieces on the board and which player should make the next move.

   The `GameState` interface specifies these methods. The `Player` class will be described in part (a).

```
public interface GameState
{
  /** @return true if the game is in an ending state;
   *          false otherwise
   */
  boolean isGameOver();


  /** Precondition: isGameOver() returns true
   *  @return the player that won the game or null if there was no winner
   */
  Player getWinner();


  /** Precondition: isGameOver() returns false
   *  @return the player who is to make the next move
   */
  Player getCurrentPlayer();


  /** @return a list of valid moves for the current player;
   *          the size of the returned list is 0 if there are no valid moves.
   */
  ArrayList<String> getCurrentMoves();


  /** Updates game state to reflect the effect of the specified move.
   *  @param move a description of the move to be made
   */
  void makeMove(String move);


  /** @return a string representing the current GameState
   */
  String toString();

}
```

   The `makeMove` method makes the move specified, updating the state of the game being played. Its parameter is a `String` that describes the move. The format of the string depends on the game. In tic-tac-toe, for example, the move might be something like `"X-1-1"`, indicating an X is put in the position (1, 1).

AP® COMPUTER SCIENCE A
2007 SCORING GUIDELINES

Question 3: Answer Sheets

Part A: getScore **4 1/2 points**

+1/2 initialize score (a double) or right/wrong counters

+1 1/2 loop over either `answers` or `key`
- **+1/2** reference `answers` or `key` in loop body
- **+1/2** correctly access `answers` or `key` element in loop body
- **+1/2** access all `answers` or `key` elements

+2 calculate score
- **+1/2** attempt to compare an `answers` element and a `key` element (== ok)
- **+1/2** correctly compare corresponding elements using `equals`
- **+1/2** add 1 to score if and only if equal
- **+1/2** subtract 1/4 from score if and only if not equal and answer not "?"

+1/2 return calculated score

Part B: highestScoringStudent **4 1/2 points**

+1 1/2 loop over `sheets`
- **+1/2** reference `sheets` in loop body
- **+1/2** correctly access `sheets` element in context of loop
- **+1/2** access all elements of `sheets`

+2 determine highest score
- **+1/2** get student score (call `getScore(key)` on a `sheets` element)
- **+1/2** compare student score with highest so far (in context of loop)
- **+1** correctly identify highest score (lose this if use constant for initial high)

+1 return name
- **+1/2** access name (call `getName` on highest)
- **+1/2** return name

AP® COMPUTER SCIENCE A
2007 CANONICAL SOLUTIONS

Question 3: Answer Sheets

PART A:

```
public double getScore(ArrayList<String> key)
{
    double score = 0.0;
    for (int i = 0; i < answers.size(); i++) {
        if (answers.get(i).equals(key.get(i))) {
            score += 1.0;
        }
        else if (!answers.get(i).equals("?")) {
            score -= 0.25;
        }
    }
    return score;
}
```

PART B:

```
public String highestScoringStudent(ArrayList<String> key)
{
    StudentAnswerSheet highest = sheets.get(0);
    for (StudentAnswerSheet sheet : sheets) {
        if (sheet.getScore(key) > highest.getScore(key)) {
            highest = sheet;
        }
    }
    return highest.getName();
}
```