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
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();
    }
(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 $\frac{1}{4}$ of a point is deducted for each incorrect answer. Omitted answers (indicated by "?") do not change the student's score.

Complete method `getScore` below.

```java
/**
 * @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)
{
    double score = 0.0;
    for (int num = 0; num < key.size(); num++)
    {
        if (answers.get(num).equals(key.get(num)))
            score += 1.0;
        else if (!answers.get(num).equals("?"))
            score -= 0.25;
    }
    return score;
}
```

Part (b) begins on page 14.
Complete method `highestScoringStudent` below.

```java
/** Precondition: sheets.size() > 0;
 * @param key the list of correct answers represented as strings of length one
 * Precondition: key.size() is equal to the number of answers
 * @return the name of the student with the highest score
 */
public String highestScoringStudent(ArrayList<String> key)
{
    double highScore = sheets.get(0).getScore(key);
    String highName = sheets.get(0).getName();
    for (int n = 1; n < sheets.size(); n++)
    {
        double score = sheets.get(n).getScore(key);
        if (score > highScore)
        {
            highScore = score;
            highName = sheets.get(n).getName();
        }
    }
    return highName;
}
```
(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 1/4 of a point is deducted for each incorrect answer. Omitted answers (indicated by "?") do not change the student's score.

Complete method `getScore` below.

```java
/** *
 * @param key the list of correct answers, represented as strings of length one
 * @return this student's test score
 */
public double getScore(ArrayList<String> key)
{
    double totalScore = 0.0;
    for (int i = 0; i < key.size(); i++)
    {
        if (key.get(i) == answers.get(i))
            totalScore += 1.0;
        if (key.get(i) != answers.get(i))
        {
            if (answers.get(i) == "?")
                totalScore += 0.0;
            else
                totalScore -= 0.25;
        }
    }
    return totalScore;
}
```

Part (b) begins on page 14.
Complete method highestScoringStudent below.

```java
/**  
  * 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
  * @return the name of the student with the highest score
  */
public String highestScoringStudent(ArrayList<String> key)
    return key.get(max);
```
(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.

```java
/** @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)
{
    double score = 0;
    for (k = 0; k < answers.length; k++)
    {
        if (answer[k] == key)
        {
            score += 1;
        } else
        {
            score -= 0.25;
        }
    }
    return score;
}
```

Part (b) begins on page 14.
Complete method `highestScoringStudent` below.

```java
/**  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) {
    
```
Overview

This question concerned abstraction, ArrayList traversal, and the application of basic algorithms. Students were provided with the framework of a StudentAnswerSheet class, which represents a sequence of answers to a multiple-choice test. In part (a) students were required to implement the getScore method, which takes an answer key (an ArrayList of Strings) and determines the score for the given answer sheet. This involved traversing both ArrayLists (the answer sheet and the key), comparing the strings at corresponding indices, and assigning points based on whether they matched. In part (b) the framework for a client class was provided, which stores an ArrayList of answer sheets as a field. Students were required to implement the highestScoringStudent method, which finds and returns the name of a student from the ArrayList with highest score. This involved traversing the ArrayList, calling the getScore method from part (a) on each answer sheet, identifying the sheet with highest score, accessing the name associated with that sheet, and returning that name.

Sample: A3a
Score: 9

In part (a) the student correctly initializes score, a variable of type double, to 0.0.

In the body of a properly constructed loop, each element of the ArrayList answers is compared with the corresponding element of the ArrayList key. The elements are accessed using the ArrayList get() method, and the comparison is made correctly using the String equals() method. The solution correctly increments the accumulated score by 1 if and only if the answers and key elements are identical. It also correctly decreases the accumulated score by 0.25 if and only if the answers and key elements differ, and the answers element is not a "?". As required, the accumulated score is not updated when the answers element is "?" to indicate an omitted response.

The calculated score is returned.

This part earned the maximum 4½ points.

In part (b) the solution uses variables highScore and highName to keep track of the highest score and corresponding name. Both highScore and highName are appropriately initialized from the state information of the first answer sheet (sheets.get(0)).

In the body of a properly constructed loop, the score of each answer sheet is correctly extracted using the method getScore(key) and is compared against the current value of highScore. When the score from the answer sheet is greater than the current highScore value, highScore is reassigned the answer sheet score, and highName is reassigned the name associated with that answer sheet.

The method correctly returns the name from an answer sheet with the highest score.

This part earned the maximum 4½ points.
Question 3 (continued)

Sample: A3b
Score: 6

The solution for part (a) correctly initializes totalScore, a variable of type double, to 0.0. In the body of a properly constructed loop, each element of the ArrayList answers is compared with the corresponding element of the ArrayList key. The elements are correctly accessed using the ArrayList get() method. This solution lost a ½ point because the comparison is attempted using == rather than the String equals() method. The solution correctly increments the accumulated totalScore by 1 if and only if the corresponding answers and key elements are identical. It also correctly decreases the accumulated score by 0.25 if and only if the corresponding answers and key elements differ, and the answers element is not a “?”. As required, the accumulated totalScore is not modified when the answers element is “?”, which is used to indicate an omitted response.

The calculated score is returned.

Part (a) of this solution earned 4 out of 4½ possible points.

Part (b) of this solution uses a double variable, max, to keep track of the highest score. An attempt is made to initialize max with the score from the first answer sheet.

A properly constructed loop sequences through the elements of sheets, correctly selecting elements using the get() method and accessing exactly all of the elements of sheets, so three ½ points were earned. There is no attempt to access the score of an answer sheet within the loop, however, and the reference to getScore() in the initialization is missing the key parameter. So the solution lost this ½ point. A comparison is attempted, but it is between a sheets element (sheets.get(i)) and a score (max); further, the update attempts to assign max (a double) from sheets.get(i) (a StudentAnswerSheet). Consequently, this solution lost the full 1 point for correctly identifying the highest score but was awarded the ½ point for attempting to compare a student score with the highest so far.

There is no attempt to access or return a name, so both ½ points for returning the name of the student with the highest score were lost.

This part of the solution earned 2 out of 4½ possible points.
Sample: A3c
Score: 3

Part (a) of this solution correctly initializes `score`, a variable of type `double`, to 0.0.

The loop whose purpose is to sequence through the elements of `answers` and/or `key` has an inappropriate test, `>=`. Further, although there are attempts to reference elements of both `answers` and `key` within the loop, the indexing mechanism used is incorrect for an `ArrayList`. Consequently, the solution was awarded a ½ point for looping over `answers` and/or `key` but lost the ½ point for correct access using `get()` and the ½ point for accessing exactly all of the elements. This solution was awarded the ½ point for attempting to compare `answers` and `key` elements but lost the ½ point for doing so correctly using the `equals()` method. The solution exhibits the logic required to increment the score by 1 for a correct answer and to decrement the score by 0.25 for an incorrect answer, and makes no adjustment for an omitted answer. It was awarded both ½ points for correctly updating the score.

This solution earned the ½ point allocated for correctly returning a calculated score.

This part earned 3 out of 4½ possible points.

The student made no attempt at part (b), so no points were earned for this part of the question.