AP Computer Science A
Sample Student Responses and Scoring Commentary

Inside:

Free Response Question 2
☑ Scoring Guideline
☑ Student Samples
☑ Scoring Commentary
Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion ([] get)

w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

x) Local variables used but none declared

y) Destruction of persistent data (e.g., changing value referenced by parameter)

z) Void method or constructor that returns a value

No Penalty

o Extraneous code with no side-effect (e.g., valid precondition check, no-op)

o Spelling/case discrepancies where there is no ambiguity*

o Local variable not declared provided other variables are declared in some part

o private or public qualifier on a local variable

o Missing public qualifier on class or constructor header

o Keyword used as an identifier

o Common mathematical symbols used for operators (× • ÷ < > ≠)

o [] vs. () vs. <>

= instead of == and vice versa

o length/size confusion for array, String, List, or ArrayList; with or without ( )

o Extraneous [] when referencing entire array

o [i,j] instead of [i][j]

o Extraneous size in array declaration, e.g., int[size] nums = new int[size];

o Missing ; where structure clearly conveys intent

o Missing {} where indentation clearly conveys intent

o Missing () on parameter-less method or constructor invocations

o Missing () around if or while conditions

*Spelling and case discrepancies for identifiers fall under the “No Penalty” category only if the correction can be unambiguously inferred from context, for example, “ArayList” instead of “ArrayList”. As a counterexample, note that if the code declares “int G=99, g=0;”, then uses “while (G < 10)” instead of “while (g < 10)”, the context does not allow for the reader to assume the use of the lower case variable.
Question 2: Word Pair

<table>
<thead>
<tr>
<th>Part (a)</th>
<th>WordPairList</th>
<th>5 points</th>
</tr>
</thead>
</table>

**Intent:** Form pairs of strings from an array and add to an ArrayList

- +1 Creates new ArrayList and assigns to allPairs
- +1 Accesses all elements of words (no bounds errors)
- +1 Constructs new WordPair using distinct elements of words
- +1 Adds all necessary pairs of elements from word array to allPairs
- +1 **On exit:** allPairs contains all necessary pairs and no unnecessary pairs

<table>
<thead>
<tr>
<th>Part (b)</th>
<th>numMatches</th>
<th>4 points</th>
</tr>
</thead>
</table>

**Intent:** Count the number of pairs in an ArrayList that have the same value

- +1 Accesses all elements in allPairs (no bounds errors)
- +1 Calls getFirst or getSecond on an element from list of pairs
- +1 Compares first and second components of a pair in the list
- +1 Counts number of matches of pair-like values

**Question-Specific Penalties**

- -1 (2) Constructor returns a value
## Question 2: Scoring Notes

### Parts

#### Part (a) WordPairList

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point if they...</th>
<th>Responses will not earn the point if they...</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Creates new ArrayList and assigns to allPairs</td>
<td>allPairs = new ArrayList(); allPairs = new ArrayList&lt;&gt;(); this.allPairs = ...</td>
<td>initialize a local variable that is never assigned to allPairs</td>
</tr>
<tr>
<td>+1</td>
<td>Accesses all elements of words (no bounds errors)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Constructs new WordPair using distinct elements of words</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Adds all necessary pairs of elements from word array to allPairs</td>
<td>have a loop bounds error; add unnecessary pairs</td>
<td>improperly add to an ArrayList, e.g., allPairs.get(i) = x; only add consecutive pairs (words[i], words[i+1])</td>
</tr>
<tr>
<td>+1</td>
<td>On exit: allPairs contains all necessary pairs and no unnecessary pairs</td>
<td>improperly add to an ArrayList, e.g., allPairs.get(i) = x; have a loop bounds error</td>
<td>add pairs (i, i) or (i, j) where i &gt; j</td>
</tr>
</tbody>
</table>

#### Part (b) numMatches

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point if they...</th>
<th>Responses will not earn the point if they...</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Accesses all elements in allPairs (no bounds errors)</td>
<td></td>
<td>access elements of allPairs as array elements (e.g., allPairs[i])</td>
</tr>
<tr>
<td>+1</td>
<td>Calls getFirst or getSecond on an element from list of pairs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Compares first and second components of a pair in the list</td>
<td></td>
<td>compare using ==</td>
</tr>
<tr>
<td>+1</td>
<td>Counts number of matches of pair-like values</td>
<td></td>
<td>fail to initialize the counter</td>
</tr>
</tbody>
</table>

Return is not assessed in part (b).
Question 2: Word Pair

Part (a)

```java
public WordPairList(String[] words) {
    allPairs = new ArrayList<WordPair>();

    for (int i = 0; i < words.length - 1; i++)
        for (int j = i + 1; j < words.length; j++)
            allPairs.add(new WordPair(words[i], words[j]));
}
```

Part (b)

```java
public int numMatches() {
    int count = 0;

    for (WordPair pair: allPairs)
        if (pair.getFirst().equals(pair.getSecond()))
            count++;

    return count;
}
```
Complete the WordPairList constructor below.

```java
/** Constructs a WordPairList object as described in part (a).
 * Precondition: words.length >= 2
 */
public WordPairList(String[] words) {
    ArrayList<WordPair> p = new ArrayList<WordPair>();
    for (int i = 0; i < words.length - 1; i++)
        for (int j = i + 1; j < words.length; j++)
            p.add(new WordPair(words[i], words[j]));
    allPairs = p;
}
```
Complete method numMatches below.

```java
/** Returns the number of matches as described in part (b).
 */
public int numMatches() {
    int count = 0;
    for (int i = 0; i < allPairs.size(); i++) {
        WordPair temp = allPairs.get(i);
        if ((temp.getFirst()).equals(temp.getSecond()))
            count++;
    }
    return count;
}
```
Complete the `WordPairList` constructor below.

```java
/** Constructs a `WordPairList` object as described in part (a).
 * Precondition: `words.length >= 2`
 */
public WordPairList(String[] words)
{
    ArrayList<WordPair> list = new ArrayList<>();
    for (i = 0; i < words.length; i++)
        for (k = 1; k < words.length; k++)
            WordPair temp = new WordPair(words[i], words[k]);
            list.add(temp);
    for (WordPair pairs : list)
        allPairs.add(pairs);
}
```

Part (b) begins on page 12
Complete method numMatches below.

```java
/** Returns the number of matches as described in part (b).
 */
public int numMatches()
{
    int matches = 0;
    for (int i = 0; i < allPairs.size(); i++)
        if (allPairs.get(i).getFirst().equals(allPairs.get(i).getSecond()))
            matches++;
    return matches;
}
```

2Bb
Complete the `WordPairList` constructor below.

```java
/** Constructs a `WordPairList` object as described in part (a).
 * Precondition: `words.length >= 2`
 */
public WordPairList(String[] words)
{
    for (int k = 0; k < words.length; k++)
    {
        for (int j = k+1; j < words.length; j++)
        {
            allPairs.add(new WordPair(k, j));
        }
    }
}
```

Part (b) begins on page 12
Complete method `numMatches` below.

```java
    /** Returns the number of matches as described in part (b). 
     */
    public int numMatches()
    {
        int numMatches = 0;

        for (int i = 0; i < allPairs.size(); i++)
        {
            for (int j = 0; j < allPairs.size(); j++)
            {
                if (i != j)
                {
                    if (allPairs.get(i).equals(allPairs.get(j))
                        numMatches++;
                }
            }
        }

        return numMatches;
    }
```

GO ON TO THE NEXT PAGE.
Question 2

Overview

This question tested the student’s ability to:

- Write program code to define a new type by creating a class;
- Write program code to create objects of a class and call methods; and
- Write program code to create, traverse, and manipulate elements in 1D array or ArrayList objects.

Students were asked to write a constructor and a method of the WordPairList class. In writing the constructor, students were expected to access an array of strings in order to populate an ArrayList of WordPair objects. Students were also expected to traverse the list of objects in order to count how many elements met a specified requirement. A provided WordPair class is used to represent pairs of words extracted from the array.

In part (a) the students were asked to write a constructor for the WordPairList class. Students needed to recognize that the ArrayList instance variable must be constructed before elements can be added. To populate the list, the students were expected to write a loop structure to pair each element from the words array parameter with each of the subsequent elements in the array. Students were expected to construct a WordPair object from each of the paired elements and add each WordPair object to the allPairs instance variable.

In part (b) students were expected to access all the elements of allPairs to count how many WordPair elements consisted of pairs of matching strings. Students were expected to call the WordPair methods getFirst and getSecond to access each word component in the pair. To compare the words, students were expected to use appropriate methods of the String class, such as equals or compareTo. To count the number of matching words, students were expected to declare and initialize an accumulator before their loop structure and increment it only when a match was found. While the method was required to return a value, the return of the accumulator was not assessed in this question.

Sample: 2A
Score: 9

In part (a) the response earned point 1 because it creates a new ArrayList as a local variable and then assigns it to allPairs after populating the list. Point 2 was earned by accessing all elements of the words array in the nested for loops with no bounds errors. The response earned point 3 by correctly constructing a new WordPair object using two distinct elements of words. Points 4 and 5 were earned by correctly adding all necessary pairs of elements from the word array to allPairs. Within the nested loops, the inner loop control variable is initialized to be one greater than the outer loop control variable, which pairs each element with all subsequent elements of the word array and prevents the addition of unnecessary pairs. Part (a) earned 5 points.

In part (b) point 6 was earned by correctly accessing all elements using a for loop and the get method on allPairs. Point 7 was earned by correctly calling the methods getFirst and getSecond on elements of allPairs. The response compares the first and second components of a pair in the list using the equals method, which earned point 8. Point 9 was earned by correctly initializing a count variable and incrementing it only when a match is found. Part (b) earned 4 points.
AP® COMPUTER SCIENCE A
2018 SCORING COMMENTARY

Question 2 (continued)

Sample: 2B
Score: 6

In part (a) the response creates a new ArrayList as a local variable and copies all elements to allPairs. Because the instance variable allPairs is never instantiated, the response did not earn point 1. Point 2 was earned by accessing all elements of the words array in the nested for loops. Although the response constructs a WordPair object, point 3 was not earned because the second parameter is an index instead of an element of words. Point 4 was earned by correctly adding all necessary pairs to allPairs. Within the nested loops, the inner loop control variable is initialized to 1, which pairs each element with all elements except the first element of the word array. Because of this, the response adds unnecessary pairs in which the first component of the pair occurs after the second component in the array of words, and it did not earn point 5. Part (a) earned 2 points.

In part (b) point 6 was earned by correctly accessing all elements using a for loop and the get method on allPairs. The response compares the first and second components of a pair in the list using the equals method, which earned point 8. Point 9 was earned by correctly initializing a count variable and incrementing it only when a match is found. Even though the response does not close the brace of the for loop, the indentation demonstrates that the return statement is located outside the loop. Part (b) earned 4 points.

Sample: 2C
Score: 2

In part (a) the response did not earn point 1 because it does not create and assign a new ArrayList to allPairs. Point 2 was not earned because the response does not access any elements of the array words. Incorrect use of the size method to obtain the length of the array did not affect the response’s score. Because the construction of the WordPair object is missing the keyword new and the parameters are indexes instead of elements of words, point 3 was not earned. Similarly, point 4 was not earned because the response adds pairs of indexes to the list instead of adding pairs of elements of the word array. Point 5 was earned by initializing the inner loop control variable to be one greater than the outer loop control variable, ensuring that no unnecessary pairs are added to the list. Part (a) earned 1 point.

In part (b) point 6 was earned by correctly accessing all elements using a for loop and the get method on allPairs. Because the response does not call the getFirst or getSecond methods, point 7 was not earned. Point 8 was not earned because the response compares WordPair objects instead of components of the word pair. Point 9 was not earned because the condition that guards the accumulator compares different pairs instead of portions of the same pair; therefore, pair-like values are not being counted. Part (b) earned 1 point.