AP® COMPUTER SCIENCE A
2016 GENERAL SCORING GUIDELINES

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., writing to output, failure to compile)

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

- Extraneous code with no side-effect (e.g., precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. <>
- = instead of == and vice versa
- `length`/`size` confusion for array, String, List, or ArrayList; with or without ( )
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing ; where structure clearly conveys intent
- Missing { } where indentation clearly conveys intent
- Missing ( ) on parameter-less method or constructor invocations
- 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 counter example, note that if the code declares* "Bug bug;"*, then uses* "Bug.move()" *instead of* "bug.move()"*, the context does **not** allow for the reader to assume the object instead of the class.*

AP® COMPUTER SCIENCE A
2016 SCORING GUIDELINES

Question 4: String Formatter

Part (a)	`totalLetters`	**2 points**

Intent: *Calculate the total number of letters in a list of words*

+1 Accesses all strings in `wordList` and adds length of each to accumulated count (*no bounds errors*)

+1 Initializes and returns accumulated count

Part (b)	`basicGapWidth`	**2 points**

Intent: *Calculate the minimum number of spaces (basic gap width) to be placed between each word in the formatted string*

+1 Calls `totalLetters` correctly and uses result

+1 Returns correct calculated value

Part (c)	`format`	**5 points**

Intent: *Return a formatted string consisting of words from `wordList` separated by one or more spaces*

+1 Calls `basicGapWidth` and `leftoverSpaces` correctly and uses results

+1 Adds all strings in `wordList` to formatted string in original order (*no bounds errors*)

+1 Inserts *basicGapWidth* spaces between each pair of words in formatted string

+1 Inserts one space between first *leftoverSpaces* pairs of words in formatted string

+1 Initializes and returns formatted string (*no extra or deleted characters*)

AP® COMPUTER SCIENCE A
2016 CANONICAL SOLUTIONS

Question 4: String Formatter

Part (a):

```
public static int totalLetters(List<String> wordList)
{
      int total = 0;

      for (String word : wordList)
      {
            total += word.length();
      }
      return total;
}
```

Part (b):

```
public static int basicGapWidth(List<String> wordList, int formattedLen)
{
      return (formattedLen – totalLetters(wordList)) / (wordList.size()-1);
}
```

Part (c):

```
public static String format(List<String> wordList, int formattedLen)
{
      String formatted = "";
      int gapWidth = basicGapWidth(wordList, formattedLen);
      int leftovers = leftoverSpaces(wordList, formattedLen);

      for (int w = 0; w < wordList.size() - 1; w++)
      {
            formatted = formatted + wordList.get(w);
            for (int i = 0; i < gapWidth; i++)
            {
                  formatted = formatted + " ";
            }
            if (leftovers > 0)
            {
                  formatted = formatted + " ";
                  leftovers--;
            }
      }
      formatted = formatted + wordList.get(wordList.size() - 1);

      return formatted;
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

4Aa

(a) Write the `StringFormatter` method `totalLetters`, which returns the total number of letters in the words in its parameter `wordList`. For example, if the variable `List<String> words` is `["A", "frog", "is"]`, then the call `StringFormatter.totalLetters(words)` returns 7. You may assume that all words in `wordList` consist of one or more letters.

Complete method `totalLetters` below.

```
/** Returns the total number of letters in wordList.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 */
public static int totalLetters(List<String> wordList){
    int total = 0;
    for(int i = 0; i<wordList.size(); i++){
        int leng = wordList.get(i).length();
        total += leng;
    }
    return total;
}
```

Part (b) begins on page 22.

4Ab

Assume that `totalLetters` works as specified regardless of what you wrote in part (a). You must use `totalLetters` appropriately to receive full credit.

Complete method `basicGapWidth` below.

```
/** Returns the basic gap width when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
{
    int letters = totalLetters(wordList);
    int gaps = wordList.size() - 1;
    int base = formattedLen - letters;
    int ans = base/gaps;
    return ans;
}
```

Part (c) begins on page 24.

4Ac

Assume that `basicGapWidth` works as specified, regardless of what you wrote in part (b). You must use `basicGapWidth` and `leftoverSpaces` appropriately to receive full credit.

Complete method `format` below.

```
/** Returns a formatted string consisting of the words in wordList separated by spaces.
 *  Precondition: The wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 *  Postcondition: All words in wordList appear in the formatted string.
 *    - The words appear in the same order as in wordList.
 *    - The number of spaces between words is determined by basicGapWidth and the
 *      distribution of leftoverSpaces from left to right, as described in the question.
 */
public static String format(List<String> wordList, int formattedLen)
    String s = "";
    int numGaps = wordList.size() - 1;
    int base = basicGapWidth(wordList, formattedLen);
    int extra = leftoverSpaces(wordList, formattedLen);
    int k=0,
    for(int i=0; i<wordList.size(); i++){

        s += wordList.get(i)
    int m=0
        while (m<base)
            s += " ";
            m++;
        }
        if(k<extra){
            s+=" ";
            k++,
        }

    }
    return s;

}
```

4Ba

(a) Write the `StringFormatter` method `totalLetters`, which returns the total number of letters in the words in its parameter `wordList`. For example, if the variable `List<String> words` is `["A", "frog", "is"]`, then the call `StringFormatter.totalLetters(words)` returns 7. You may assume that all words in `wordList` consist of one or more letters.

Complete method `totalLetters` below.

```
/** Returns the total number of letters in wordList.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 */
public static int totalLetters(List<String> wordList)
{
    String all = "";
    for ( int i = 0; i < wordList.size(); i++ )
    {
        all += wordList.get(i);
    }
    return all.length();
}
```

Part (b) begins on page 22.

Assume that `totalLetters` works as specified regardless of what you wrote in part (a). You must use `totalLetters` appropriately to receive full credit.

Complete method `basicGapWidth` below.

```
/** Returns the basic gap width when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
{
    int letters = wordList.totalLetters(wordList);
    int width = (formattedLen - letters) / (wordList.size() - 1);
    return width;
}
```

Part (c) begins on page 24.

4Bc

Assume that `basicGapWidth` works as specified, regardless of what you wrote in part (b). You must use `basicGapWidth` and `leftoverSpaces` appropriately to receive full credit.

Complete method `format` below.

```
/** Returns a formatted string consisting of the words in wordList separated by spaces.
 *  Precondition: The wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 *  Postcondition: All words in wordList appear in the formatted string.
 *    - The words appear in the same order as in wordList.
 *    - The number of spaces between words is determined by basicGapWidth and the
 *      distribution of leftoverSpaces from left to right, as described in the question.
 */
public static String format(List<String> wordList, int formattedLen)
{
    String all = "";
    for(int i = 0; i < wordList.size(); i++)
    {
        all += wordList.get(i);
        for(int j = 0; j < basicGapWidth(wordList, formattedLen); j++)
        {
            all += " ";
        }
    }

    return all;
}
```

4Ca

(a) Write the `StringFormatter` method `totalLetters`, which returns the total number of letters in the words in its parameter `wordList`. For example, if the variable `List<String> words` is `["A", "frog", "is"]`, then the call `StringFormatter.totalLetters(words)` returns 7. You may assume that all words in `wordList` consist of one or more letters.

Complete method `totalLetters` below.

```
/** Returns the total number of letters in wordList.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 */
public static int totalLetters(List<String> wordList)
```

```
for ( int k = 0; k < wordlist.length; k++ )
    { total += wordlist.get(k).length; }
    return total;
```

Part (b) begins on page 22.

4Cb

Assume that `totalLetters` works as specified regardless of what you wrote in part (a). You must use `totalLetters` appropriately to receive full credit.

Complete method `basicGapWidth` below.

```
/** Returns the basic gap width when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
{  int basicGapWidth = 0;
   basicGapWidth = (formattedLen - wordList.totalLetters)/
                   (wordList.length - 1)

}
```

Part (c) begins on page 24.

4Cc

Assume that `basicGapWidth` works as specified, regardless of what you wrote in part (b). You must use `basicGapWidth` and `leftoverSpaces` appropriately to receive full credit.

Complete method `format` below.

```
/** Returns a formatted string consisting of the words in wordList separated by spaces.
 *  Precondition: The wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 *  Postcondition: All words in wordList appear in the formatted string.
 *    - The words appear in the same order as in wordList.
 *    - The number of spaces between words is determined by basicGapWidth and the
 *      distribution of leftoverSpaces from left to right, as described in the question.
 */
public static String format(List<String> wordList, int formattedLen)
```

```
String formatted = " ";

for (int k = 0; k < wordList.length; k++)
    for (int j = 0; j < wordList.basicGapWidth(); j++)
        formatted = wordList(k) + " "
```

formated

AP® COMPUTER SCIENCE A
2016 SCORING COMMENTARY

Question 4

Overview

This question assessed the students' ability to use `ArrayLists`, `Strings`, and mathematical expressions involving integers. Students also needed to understand how to use iteration and proper method calls. Students were provided with specifications for three `static` methods that would be used to create a properly spaced string of a specified length that contained a set of words given in an `ArrayList` object, `wordList`.

In part (a) students were asked to write the `totalLetters` method that sums and returns the lengths of all of the strings in its parameter `wordList`.

In part (b) students were asked to write a method to compute the minimum number of spaces necessary between each pair of words to space them out to the given length.

In part (c) students were asked to write the method that creates a single string of the given length by inserting spaces into the gaps between words so that the number of spaces in each gap is the same. If more spaces were needed to reach the given length, an additional space would be added to each gap, starting with the first gap, so that the lengths of the gaps would differ by at most 1. In doing this, they were required to call the `basicGapWidth` method written in part (b), as well as the `leftoverSpaces` method described in the problem but not implemented by the student.

To solve this problem, the student needed to traverse an `ArrayList` with no bounds errors, using the `get` method of the `ArrayList` class in an indexed `for` loop. The student could use an enhanced `for` (for-each) loop, but needed to understand the consequences of not having an index to manipulate in part (c). The student needed to properly implement a `static` method that initialized, accumulated, and returned a computed value. The student needed to properly determine the length of a string, as well as initialize an empty string and concatenate strings.

The student needed to properly call methods from within a class, including those they have implemented, and those for which they have only a description. Students needed to translate a mathematical expression into an appropriate Java expression, and understand and implement a complex algorithm from a written description.

Sample: 4A
Score: 8

In part (a) the student writes a correct `for` loop to access every string in `wordList` and sums the lengths of all the strings into the accumulator `total`, which earned the "accesses all" point. The accumulator is properly declared, initialized, and returned, which earned the "initialize and return" point. Part (a) earned 2 points.

In part (b) the student correctly calls `totalLetters(wordList)` and saves the result into a variable. This variable is used in a subsequent calculation, which earned the "calls `totalLetters`" point. The result is calculated correctly and returned, which earned the "return correct value" point. Part (b) earned 2 points.

In part (c) the student correctly calls `basicGapWidth` and `leftOverSpaces`, saving the results in variables. These variables are used in subsequent calculations, which earned the "calls methods" point. The student uses a `for` loop to properly access each string in `wordList` and adds each to the result string, which earned the "adds all" point. By initializing a counter to 0 and using a `while` loop with an increment,

Question 4 (continued)

the student adds *basicGapWidth* spaces to the string between each pair of words and adds single spaces to the first *leftoverSpaces* gaps, which earned the "*basicGapWidth*" and "*leftoverSpaces*" points. Because *basicGapWidth* spaces are added after every word including the last, instead of only in the gaps between words, the final returned string is incorrect, and the final point was not earned. Part (c) earned 4 points.

Sample: 4B
Score: 5

In part (a) the student writes a correct `for` loop to access every string in `wordList`, and concatenates these strings into one long string whose length is returned, which earned the "accesses all" point. The long string, which plays the role of the accumulator in this solution, is properly declared, initialized, and returned, which earned the "initialize and return" point. Part (a) earned 2 points.

In part (b) the student incorrectly calls `totalLetters` as if it were a method of the `ArrayList` class, and so the "calls `totalLetters`" point was not earned. The calculation is correct, and the result is returned, which earned the "return correct value" point. Part (b) earned 1 point.

In part (c) the student correctly calls `basicGapWidth` and uses its result, but does not call `leftoverSpaces`, so the "calls methods" point was not earned. The student uses a `for` loop to correctly access all strings in `wordList`, which earned the "adds all" point. The inner `for` loop correctly adds *basicGapWidth* spaces to the result string after each word, which earned the "*basicGapWidth*" point. Because there is no attempt to call `leftoverSpaces`, the "*leftoverSpaces*" point was not earned. Because *basicGapWidth* spaces are added after every word including the last, instead of only in the gaps between words, the final returned string is incorrect, and the final point was not earned. Part (c) earned 2 points.

Sample: 4C
Score: 1

In part (a) the student uses a `for` loop to add the lengths of all strings in `wordList`. Use of the incorrect method `length` instead of `size` in the loop condition is not penalized, so the student earned the "accesses all" point. The accumulator is not initialized, so the "initialize and return" point was not earned. Part (a) earned 1 point.

In part (b) the student attempts to use a field called `totalLetters` instead of making a method call, so the "calls `totalLetters`" point was not earned. The calculation is incorrect, and its result is not returned, so the "return correct value" point was not earned. Part (b) did not earn any points.

In part (c) the student makes an incorrect call to `basicGapWidth` and makes no attempt to call `leftOverSpaces`, so the "calls methods" point was not earned. The student uses a `for` loop to correctly access each element of `wordList`, but simply assigns the element to the formatted string during each iteration instead of appending it to the formatted string, so the "adds all" point was not earned. Because of this error, *basicGapWidth* spaces are not added to the formatted string, so the "*basicGapWidth*" point was not earned. There is no attempt to add an additional space to each of the first *leftOverSpaces* gaps, so the "*leftoverSpaces*" point was not earned. The resulting formatted string is not returned, so the final point is not earned. Part (c) did not earn any points.