AP® COMPUTER SCIENCE A
2014 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.

1-Point Penalty

(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
- Array/collection access confusion (`[] get`)
- `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 `;` provided majority are present and indentation clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent and `{ }` are used elsewhere
- 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 counterexample, 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
2014 SCORING GUIDELINES

Question 1: Word Scramble

Part (a)	`scrambleWord`	**5 points**

Intent: *Scramble a word by swapping all letter pairs that begin with A*

+1 Accesses all letters in `word`, left to right (*no bounds errors*)

+1 Identifies at least one letter pair consisting of "A" followed by non-"A"

+1 Reverses identified pair in constructing result string

+1 Constructs correct result string (*Point lost if any letters swapped more than once, minor loop bounds errors ok*)

+1 Returns constructed string

Part (b)	`scrambleOrRemove`	**4 points**

Intent: *Modify list by replacing each word with scrambled version and removing any word unchanged by scrambling*

+1 Accesses all words in `wordList` (*no bounds errors*)

+1 Calls `scrambleWord` with a word from the list as parameter

+1 Identifies words unchanged by scrambling

+1 On exit: List includes all and only words that have been changed by scrambling once, in their original relative order (*minor loop bounds errors ok*)

AP® COMPUTER SCIENCE A
2014 CANONICAL SOLUTIONS

Question 1: Word Scramble

Part (a):

```
public static String scrambleWord(String word){
   int current = 0;
   String result="";
   while (current < word.length()-1){
      if (word.substring(current,current+1).equals("A") &&
            !word.substring(current+1,current+2).equals("A")){
         result += word.substring(current+1,current+2);
         result += "A";
         current += 2;
      }
      else {
         result += word.substring(current,current+1);
         current++;
      }
   }
   if (current < word.length()){
      result += word.substring(current);
   }
   return result;
}
```

Part (b):

```
public static void scrambleOrRemove(List<String> wordList){
   int index = 0;
   while (index < wordList.size()){
      String word=wordList.get(index);
      String scrambled=scrambleWord(word);
      if (word.equals(scrambled)){
         wordList.remove(index);
      }
      else {
         wordList.set(index,scrambled);
         index++;
      }
   }
}
```

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.

1Aa

Complete method `scrambleWord` below.

```
/** Scrambles a given word.
 *  @param word the word to be scrambled
 *  @return the scrambled word (possibly equal to word)
 *  Precondition: word is either an empty string or contains only uppercase letters.
 *  Postcondition: the string returned was created from word as follows:
 *    - the word was scrambled, beginning at the first letter and continuing from left to right
 *    - two consecutive letters consisting of "A" followed by a letter that was not "A" were swapped
 *    - letters were swapped at most once
 */
public static String scrambleWord(String word)
{
  String result="";

  for(int i=0; i<word.length-1; i++)
  {
    if(word.substring(i,i+1).equals("A")&&(word.substring(
    i+1,i+2).equals("A")))
    {
      result= result + word.substring(i+1,i+2)+word.substring
      (i,i+1);
      i++
    }
    else
    {
      result= result+word.substring(i,i+1);
    }
  }
  result=result+word.substring(word.length-1);
  return result;
}
```

Part (b) begins on page 6.

1Ab

Assume that `scrambleWord` is in the same class as `scrambleOrRemove` and works as specified, regardless of what you wrote in part (a).

Complete method `scrambleOrRemove` below.

```
/** Modifies wordList by replacing each word with its scrambled
 *  version, removing any words that are unchanged as a result of scrambling.
 *  @param wordList the list of words
 *  Precondition: wordList contains only non-null objects
 *  Postcondition:
 *    - all words unchanged by scrambling have been removed from wordList
 *    - each of the remaining words has been replaced by its scrambled version
 *    - the relative ordering of the entries in wordList is the same as it was
 *        before the method was called
 */
public static void scrambleOrRemove(List<String> wordList)
{
  for(int i=0; i < wordList.size(); i++)
  {
    if(!(wordList.get(i).equals(scrambleWord(wordList.get(i)))))
    {
      wordList.set(i, scrambleWord(wordList.set(i)))
    }
    else
    {
      wordList.remove(i);
      i--
    }
}
```

1 Ba

Complete method `scrambleWord` below.

```
/** Scrambles a given word.
 *  @param word the word to be scrambled
 *  @return the scrambled word (possibly equal to word)
 *  Precondition: word is either an empty string or contains only uppercase letters.
 *  Postcondition: the string returned was created from word as follows:
 *    - the word was scrambled, beginning at the first letter and continuing from left to right
 *    - two consecutive letters consisting of "A" followed by a letter that was not "A" were swapped
 *    - letters were swapped at most once
 */
public static String scrambleWord(String word)
{
  for (int i = 0; i < word.length()-2; i++){
    if (word.substring(i,i+1).equals("A")){
    if (!(word.substring(i+1,i+2).equals("A"))){
      word = word.substring(0,i) +
             word.substring(i+1,i+2) +
             word.substring(i,i+1) +
             word.substring(i+2); i++;
    }
    }
  }
  /* Note- I'd use an "AND" statement instead of
     two IFs normally, but I realized I can't draw
     an ampersand.
  */
```

Part (b) begins on page 6.

Assume that `scrambleWord` is in the same class as `scrambleOrRemove` and works as specified, regardless of what you wrote in part (a).

Complete method `scrambleOrRemove` below.

```
/** Modifies wordList by replacing each word with its scrambled
 *  version, removing any words that are unchanged as a result of scrambling.
 *  @param wordList the list of words
 *  Precondition: wordList contains only non-null objects
 *  Postcondition:
 *    - all words unchanged by scrambling have been removed from wordList
 *    - each of the remaining words has been replaced by its scrambled version
 *    - the relative ordering of the entries in wordList is the same as it was
 *      before the method was called
 */
public static void scrambleOrRemove(List<String> wordList) {
    List<String> temp = new ArrayList<String>();
    for (int i = 0; i < wordList.size(); i++) {
        String unscram = wordList.remove(i);
        String scram = scrambleWord(unscram);
        if (!(scram.equals(unscram)))
            temp.add(scram);
    }
    wordList = temp;
}
```

1Ca

Complete method `scrambleWord` below.

```
/** Scrambles a given word.
 *  @param word the word to be scrambled
 *  @return the scrambled word (possibly equal to word)
 *  Precondition: word is either an empty string or contains only uppercase letters.
 *  Postcondition: the string returned was created from word as follows:
 *    - the word was scrambled, beginning at the first letter and continuing from left to right
 *    - two consecutive letters consisting of "A" followed by a letter that was not "A" were swapped
 *    - letters were swapped at most once
 */
public static String scrambleWord(String word)
String temp = "";

while (i < word.length-1) {++}
  if (word(i) == "A" && word(i+1) != "A") {
    temp = word(i)
    word(i) = word(i+1);
    word(i+1) = temp;
    i ++;
  }
  i ++
}

return word;
```

Part (b) begins on page 6.

GO ON TO THE NEXT PAGE.

1 C b

Assume that `scrambleWord` is in the same class as `scrambleOrRemove` and works as specified, regardless of what you wrote in part (a).

Complete method `scrambleOrRemove` below.

```
/** Modifies wordList by replacing each word with its scrambled
 *  version, removing any words that are unchanged as a result of scrambling.
 *  @param wordList the list of words
 *  Precondition: wordList contains only non-null objects
 *  Postcondition:
 *    - all words unchanged by scrambling have been removed from wordList
 *    - each of the remaining words has been replaced by its scrambled version
 *    - the relative ordering of the entries in wordList is the same as it was
 *      before the method was called
 */
public static void scrambleOrRemove(List<String> wordList)
```

```
Sting temp2 = "";
for (int i=0; i< wordList.size -1; i++)
{
  temp2 = wordList.get(i);
  Scramble word (wordList.get(i));
  if (temp2 = wordList.get(i));
  {
    wordList.remove(i);
  }
```

AP® COMPUTER SCIENCE A
2014 SCORING COMMENTARY

Question 1

Overview

This question asked the students to write two static methods for an unnamed class which had no instance variables. The question involved using methods of the `String` class (to get individual letters of the `String` and to build a new `String`) and methods of the `List` interface (to access elements of the `List` and to modify an existing `List`). In part (a), the students were required to implement a static method called `scrambleWord` to scramble a `String` parameter (`word`) by reversing pairs of adjacent letters of `word` whenever the pair consisted of "A" followed by a letter that was not "A". The students needed to access all letters in `word` from left to right, identifying the pairs of letters to be swapped. The identified pair could be reversed by rebuilding `word` or by constructing a new result `String`. All letters of `word` that were not involved in reversals were to be in their original positions in the result `String`, which was then returned by the method. In part (b), the students were required to implement a static method called `scrambleOrRemove`. The method had a `List` parameter (`wordList`) in which each element was a `String` of letters. The students needed to access each word in `wordList`, to correctly call the `scrambleWord` method of part (a) with the word as the argument, and to determine whether the word had been changed by scrambling. Upon exiting `scrambleOrRemove`, `wordList` was to contain only words changed by `scrambleWord` in their scrambled form. If the word had been scrambled by `scrambleWord`, the word was replaced in `wordList` by the scrambled version of the word. If the word was unchanged by `scrambleWord`, the word removed from `wordList`.

Note: This question involves static methods of a class having no instance variables. The use of `this.scrambleWord` or `this.anything` is incorrect in this question.

Sample: 1A
Score: 8

In part (a) the student correctly accesses all letters in `word`, looping through all one-letter substrings of `word`. The student correctly identifies all substrings equal to “A” and correctly identifies when the next substring is not “A”.

The student correctly reverses all pairs of substrings consisting of “A” followed by not “A” by constructing a new result string. The student correctly builds the result string, reversing pairs of letters when necessary and avoids using each “A” in more than one swap by incrementing the loop index an extra time after each reversal is made. After the loop, the student concatenates the last letter of `word` to the result string. However, this is not the correct thing to do in all cases. When the last letter of `word` is used in a reversal, it has already been put into the result string and would be duplicated by this assignment. Therefore, the student does not earn the point for constructing the correct result string. The result string is correctly returned. The response earned 4 points for part (a).

In part (b) the student correctly accesses all words in `wordList` using a for-loop. After the `remove` method call, the student decrements the index so that no words are skipped.

The student correctly calls the `scrambleWord` method using a word from `wordList` as a parameter. The scrambled word is correctly compared with the original word. If the word is not changed by `scrambleWord`, the word is removed from `wordList`. Otherwise the word is replaced by the result of scrambling the word. The response earned 4 points in part (b).

Question 1 (continued)

Sample: 1B
Score: 5

In part (a) the student uses a while-loop to access the letters in `word`. However, the bound on the loop causes the student to fail to access the next-to-last letter of `word` to check for the last possible pair. The bound should be `word.length – 1`. Therefore the point for accessing all letters in `word` is not earned.

The student correctly reverses all identified pairs of substrings consisting of "A" followed by not "A". The letter pair is identified by comparing adjacent one-letter substrings of `word`. When a pair is found, the student rebuilds `word` with the adjacent substrings in reversed order. The student correctly increments the position in `word` after a reversal is made so that the "A" is not identified for any more reversals. This prevents the "A" from bubbling to the end of `word`. The points for identifying pairs of letters, reversing the pairs of letters and correctly building the result string are earned. The student fails to return `word` at the end of the method, so the point for returning the constructed string is not earned. The response earned 3 points for part (a).

In part (b) the student constructs a new `ArrayList`, then uses a for-loop to access the words in `wordList`. However, the student uses the `remove` method inside of the loop without decrementing the loop control variable, causing a word to be skipped each time `remove` is called. The point for accessing all words in `wordList` is not earned.

The student removes and stores each original word from `wordList` and correctly calls the `scrambleWord` method to scramble it and store the result. The scrambled word is correctly compared with the original word. The points for calling the `scrambleWord` method and for identifying words changed by scrambling are earned. The student adds each scrambled word to the new `ArrayList`. At the end of the method, the student assigns this new `ArrayList` to `wordList`. This does not earn the point for `wordList` containing the correct words on exit from the method because the parameter is re-assigned but the original object is not modified. Methods from the `List` interface must be used to repopulate `wordList` with the scrambled words. The response earned 2 points in part (b).

Sample: 1C
Score: 3

In part (a) the student attempts to access the letters in `word` as if `word` were a mutable array of characters. Characters are accessed using array indexing and assignments to positions of `word` are attempted. A string is an immutable object and letters of the string may not be accessed and manipulated this way. Therefore, the point for accessing the letters of `word` is lost. Because the student attempts to construct a result string by illegally modifying `word`, the point for reversing a letter pair in the result string and the point for constructing a correct result string are not earned.

Because of the previous penalties, the comparison of incorrectly accessed letters is allowed in this case. The logic is correct for identifying at least one pair of letters consisting of "A" followed by a substring that is not "A" is earned, so this point is earned. The student attempts to do some construction on `word` for a result string and then returns the "modified" word. Therefore, the point for returning the constructed result string is earned. The response earned 2 points for part (a).

Question 1 (continued)

In part (b) the point for accessing all words in `wordList` is not earned because a word would be skipped after each call to the `remove` method and the last word in `wordList` would be skipped in all cases due to an incorrect loop bound.

The student makes a correct call to the `scrambleWord` method using a word from `wordList` as the parameter. Even though the student does not assign the returned value of this method call to any variable so that it can be used, the point for making the correct call is earned. Because the return value of `scrambleWord` is not captured, it is not compared to the original word and there is no attempt to replace the original value by the scrambled version in `wordList`. The point for identifying words unchanged by scrambling and the point for `wordList` containing the correct scrambled words on exit from the method are not earned. The response earned 1 point in part (b).