COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total score—50

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:
- Assume that the classes listed in the appendices have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

1. This question involves reasoning about strings made up of uppercase letters. You will implement two related methods that appear in the same class (not shown). The first method takes a single string parameter and returns a scrambled version of that string. The second method takes a list of strings and modifies the list by scrambling each entry in the list. Any entry that cannot be scrambled is removed from the list.

   (a) Write the method `scrambleWord`, which takes a given word and returns a string that contains a scrambled version of the word according to the following rules.

   - The scrambling process begins at the first letter of the word and continues from left to right.
   - If two consecutive letters consist of an "A" followed by a letter that is not an "A", then the two letters are swapped in the resulting string.
   - Once the letters in two adjacent positions have been swapped, neither of those two positions can be involved in a future swap.

   The following table shows several examples of words and their scrambled versions.

`word`	Result returned by `scrambleWord(word)`
"TAN"	"TNA"
"ABRACADABRA"	"BARCADABARA"
"WHOA"	"WHOA"
"AARDVARK"	"ARADVRAK"
"EGGS"	"EGGS"
"A"	"A"
""	""

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

Part (b) begins on page 4.

(b) Write the method `scrambleOrRemove`, which replaces each word in the parameter `wordList` with its scrambled version and removes any words that are unchanged after scrambling. The relative ordering of the entries in `wordList` remains the same as before the call to `scrambleOrRemove`.

The following example shows how the contents of `wordList` would be modified as a result of calling `scrambleOrRemove`.

Before the call to `scrambleOrRemove`:

	0	1	2	3	4
`wordList`	"TAN"	"ABRACADABRA"	"WHOA"	"APPLE"	"EGGS"

After the call to `scrambleOrRemove`:

	0	1	2
`wordList`	"TNA"	"BARCADABARA"	"PAPLE"

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

2. This question involves reasoning about the GridWorld case study. Reference materials are provided in the appendixes.

   A `Director` is a type of `Rock` that has the following characteristics.
   - A `Director` has an initial color of `Color.RED` and alternates between `Color.RED` and `Color.GREEN` each time it acts.
   - If the color of a `Director` is `Color.GREEN` when it begins to act, it will cause any `Actor` objects in its neighboring cells to turn 90 degrees to their right.

   Write the complete `Director` class, including the zero-parameter constructor and any necessary instance variables and methods. Assume that the `Color` class has been imported.

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.