(a) A string containing text and possibly delimiters has been split into *tokens* and stored in `String[] tokens`. Each token is either an open delimiter, a close delimiter, or a substring that is not a delimiter. You will write the method `getDelimitersList`, which returns an `ArrayList` containing all the open and close delimiters found in `tokens` in their original order.

The following examples show the contents of an `ArrayList` returned by `getDelimitersList` for different open and close delimiters and different `tokens` arrays.

Example 1

`openDel:`	`"("`
`closeDel:`	`")"`

`tokens:`	`"("`	`"x + y"`	`")"`	`" * 5"`

`ArrayList` of delimiters:	`"("`	`")"`

Example 2

`openDel:`	`"<q>"`
`closeDel:`	`"</q>"`

`tokens:`	`"<q>"`	`"yy"`	`"</q>"`	`"zz"`	`"</q>"`

`ArrayList` of delimiters:	`"<q>"`	`"</q>"`	`"</q>"`

Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `getDelimitersList` below.

```
/** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
public ArrayList<String> getDelimitersList(String[] tokens)
```

(b) Write the method `isBalanced`, which returns `true` when the delimiters are balanced and returns `false` otherwise. The delimiters are balanced when both of the following conditions are satisfied; otherwise, they are not balanced.

1. When traversing the `ArrayList` from the first element to the last element, there is no point at which there are more close delimiters than open delimiters at or before that point.
2. The total number of open delimiters is equal to the total number of close delimiters.

Consider a `Delimiters` object for which `openDel` is `"<sup>"` and `closeDel` is `"</sup>"`. The examples below show different `ArrayList` objects that could be returned by calls to `getDelimitersList` and the value that would be returned by a call to `isBalanced`.

Example 1
The following example shows an `ArrayList` for which `isBalanced` returns `true`. As tokens are examined from first to last, the number of open delimiters is always greater than or equal to the number of close delimiters. After examining all tokens, there are an equal number of open and close delimiters.

"^{"	"^{"	"}"	"^{"	"}"	"}"

Example 2
The following example shows an `ArrayList` for which `isBalanced` returns `false`.

"^{"	"}"	"</sup>"	"<sup>"

↑

When starting from the left, at this point, condition 1 is violated.

Example 3
The following example shows an `ArrayList` for which `isBalanced` returns `false`.

"</sup>"

↑

At this point, condition 1 is violated.

Example 4
The following example shows an `ArrayList` for which `isBalanced` returns `false` because the second condition is violated. After examining all tokens, there are not an equal number of open and close delimiters.

"<sup>"	"^{"	"}"

Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `isBalanced` below.

```
/** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
 *  Precondition: delimiters contains only valid open and close delimiters.
 */
public boolean isBalanced(ArrayList<String> delimiters)
```

4. The `LightBoard` class models a two-dimensional display of lights, where each light is either on or off, as represented by a Boolean value. You will implement a constructor to initialize the display and a method to evaluate a light.

```
public class LightBoard
{
    /** The lights on the board, where true represents on and false represents off.
     */
    private boolean[][] lights;

    /** Constructs a LightBoard object having numRows rows and numCols columns.
     *  Precondition: numRows > 0, numCols > 0
     *  Postcondition: each light has a 40% probability of being set to on.
     */
    public LightBoard(int numRows, int numCols)
    {  /* to be implemented in part (a) */  }

    /** Evaluates a light in row index row and column index col and returns a status
     *  as described in part (b).
     *  Precondition: row and col are valid indexes in lights.
     */
    public boolean evaluateLight(int row, int col)
    {  /* to be implemented in part (b) */  }

    // There may be additional instance variables, constructors, and methods not shown.

}
```

AP® COMPUTER SCIENCE A
2019 SCORING GUIDELINES

Question 3: Delimiters

Part (a)	`getDelimitersList`	**4 points**

Intent: *Store delimiters from an array in an* `ArrayList`

+1 Creates `ArrayList<String>`

+1 Accesses all elements in array `tokens` (*no bounds errors*)

+1 Compares strings in `tokens` with both instance variables (*must be in the context of a loop*)

+1 Adds delimiters into `ArrayList` in original order

Part (b)	`isBalanced`	**5 points**

Intent: *Determine whether open and close delimiters in an* `ArrayList` *are balanced*

+1 Initializes accumulator(s)

+1 Accesses all elements in `ArrayList delimiters` (*no bounds errors*)

+1 Compares strings in `delimiters` with instance variables and updates accumulator(s) accordingly

+1 Identifies and returns appropriate `boolean` value to implement one rule

+1 Identifies and returns appropriate `boolean` values for all cases

AP® COMPUTER SCIENCE A
2019 SCORING GUIDELINES

Question 3: Scoring Notes

Part (a) `getDelimitersList`			**4 points**
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Creates `ArrayList<String>`	• omit `<String>`	• omit the keyword `new`
+1	Accesses all elements in array `tokens` (*no bounds errors*)	• return incorrectly inside the loop	• treat `tokens` as a single string • access elements of `tokens` as if from an `ArrayList` (e.g., `tokens.get(i)`)
+1	Compares strings in `tokens` with both instance variables (*must be in the context of a loop*)	• access elements of `tokens` as if from an `ArrayList` (e.g., `tokens.get(i)`)	• use `==` for string comparison • treat `tokens` as a single string
+1	Adds delimiters into `ArrayList` in original order	• add a delimiter by accessing `tokens` incorrectly (e.g., `tokens.get(i)`)	• add a token that is not a delimiter • do not maintain the original delimiter order
Part (b) `isBalanced`			**5 points**
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Initializes accumulator(s)	• initialize inside the loop	• initialize an accumulator variable but don't update it
+1	Accesses all elements in `ArrayList delimiters` (*no bounds errors*)	• return incorrectly inside the loop	• access elements of `delimiters` as if from an array (e.g., `delimiters[i]`)
+1	Compares strings in `delimiters` with instance variables and updates accumulator(s) accordingly	• access elements of `delimiters` as if from an array (e.g., `delimiters[i]`)	• use `==` for string comparison • adjust an accumulator without a guarding condition
+1	Identifies and returns appropriate `boolean` value to implement one rule	• check for more closing delimiters (inside a loop) and return `false` • return `true` if the number of open and close delimiters is the same, and `false` otherwise (after a loop)	
+1	Identifies and returns appropriate `boolean` values for all cases	• have correct logic with the exception of a loop bounds error, accessing elements as if from an array, or using `==` for string comparison	• initialize accumulator inside a loop • fail to check for more closing delimiters inside a loop

AP® COMPUTER SCIENCE A
2019 SCORING GUIDELINES

Question 3: Delimiters

Part (a)

```
public ArrayList<String> getDelimitersList(String[] tokens)
{
   ArrayList<String> d = new ArrayList<String>();
   for (String str : tokens)
   {
      if (str.equals(openDel) || str.equals(closeDel))
      {
         d.add(str);
      }
   }
   return d;
}
```

Part (b)

```
public boolean isBalanced(ArrayList<String> delimiters)
{
   int openCount = 0;
   int closeCount = 0;

   for (String str : delimiters)
   {
      if (str.equals(openDel))
      {
         openCount++;
      }
      else
      {
         closeCount++;
      }

      if (closeCount > openCount)
      {
         return false;
      }
   }

   if (openCount == closeCount)
   {
      return true;
   }
   else
   {
      return false;
   }
}
```

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.