

Question 2: Class**9 points****Canonical solution**

```
public class SignedText
{
    private String firstName;
    private String lastName;

    public SignedText(String first, String last)
    {
        firstName = first;
        lastName = last;
    }

    public String getSignature()
    {
        String sig = "";
        if (!firstName.equals(""))
        {
            sig += firstName.substring(0, 1) + "-";
        }
        sig += lastName;
        return sig;
    }

    public String addSignature(String textStr)
    {
        String sig = getSignature();
        int index = textStr.indexOf(sig);

        if (index == -1)
        {
            return textStr + sig;
        }
        else if (index == 0)
        {
            return textStr.substring(sig.length()) + sig;
        }
        else
        {
            return textStr;
        }
    }
}
```

9 points

SignedText

Scoring Criteria		Decision Rules	
1	Declares class header: <code>class SignedText</code>	Responses will not earn the point if they • declare the class as <code>private</code> • include extraneous code outside the class • include <code>()</code> with or without parameters in the class header	1 point
2	Declares all appropriate <code>private</code> instance variable(s) and constructor initializes instance variable(s) using appropriate parameters	Responses can still earn the point even if they • create and store the signature using only one <code>String</code> instance variable Responses will not earn the point if they • declare any instance variables <code>static</code> • omit <code>private</code> in instance variable declaration • declare variables outside the class • declare an instance variable inside the constructor • fail to use either parameter • assign values to instance variables from an unknown source • assign initial value to local variable instead of instance variable, even if the local variables are declared as <code>private</code> • assign parameter(s) to variable(s) with incompatible data type • return a value from the constructor • define the constructor inside a method or define a method inside the constructor	1 point
3	Declares constructor header: <code>SignedText(String ____, String ____)</code>	Responses will not earn the point if they • declare the constructor as <code>private</code> or <code>static</code>	1 point
4	Declares method headers: <code>public String getSignature() public String addSignature(String ____)</code>	Responses will not earn the point if they • omit <code>public</code> in either method header or declare either method as something other than <code>public</code> • use incorrect method name • use incorrect return or parameter type	1 point
5	Compares first name to the empty string	Responses can still earn the point even if they • perform the comparison implicitly (e.g., by comparing the string's length to 0)	1 point

6	Determines appropriate signature string in both cases (<i>algorithm</i>)	Responses can still earn the point even if they - fail to return a value in some cases (<i>return from <code>getSignature</code> not assessed</i>) Responses will not earn the point if they - construct the correct string but return something else - define another method inside the signature method or vice versa	1 point
7	Calls <code>String</code> methods using correct syntax throughout the class	Responses can still earn the point even if they - call <code>substring</code> only one time - call <code>indexOf</code>, <code>contains</code>, <code>charAt</code>, or other appropriate methods Responses will not earn the point if they - only call <code>length</code> and/or <code>equals</code> without using other <code>String</code> methods - fail to call at least one <code>String</code> method which accesses elements of a string	1 point
8	Identifies the three required cases for <code>addSignature</code> using appropriate conditions	Responses can still earn the point even if they return an incorrect string in one or more cases	1 point
9	<code>addSignature</code> returns appropriate <code>String</code> in all three cases (<i>algorithm</i>)	Responses can still earn the point even if they - identify one or more of the three cases incorrectly, as long as they are unambiguously no-match, match-start, and match-end - implement <code>getSignature</code> incorrectly Responses will not earn the point if they - call <code>getSignature</code> incorrectly, or incorrectly implement its functionality in <code>addSignature</code> - fail to identify three distinct cases - build correct strings but assign them to cases incorrectly - print the string instead of or in addition to returning it - define another method inside <code>addSignature</code> or vice versa	1 point
Question-specific penalties			
None			

Alternate canonical:

```
public class SignedText
{
    private String signature;

    public SignedText(String first, String last)
    {
        signature = last;
        if (first.length() > 0)
        {
            signature = first.substring(0, 1) + "-" + last;
        }
    }

    public String getSignature()
    {
        return signature;
    }

    public String addSignature(String textStr)
    {
        String result = textStr;
        int index = textStr.indexOf(signature);
        if (index < 0)
        {
            result += signature;
        }
        else if (index == 0)
        {
            result = result.substring(signature.length()) + signature;
        }
        return result;
    }
}
```

Applying the Scoring Criteria

Apply the question scoring criteria 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

- Extraneous code with no side-effect (e.g., valid 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 ($\times$ • $\div$ $\leq$ $\geq$ $<>$ $\neq$)
- `[]` 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, “ArrayList” 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.*

2025 Digital Decision Rules

- Some non-ASCII characters are not printing correctly in ONE, particularly extended punctuation. Certain kinds of double-quotes may display as â[?] [?]; a character that displays as ï¼[?] may be a parenthesis, comma, or semicolon (or other punctuation). If the badly displayed character makes sense as one of those, evaluate the response accordingly.
- If there are missing closed-double-quotes or closed-parentheses, assume they are at the end of the line where they opened, immediately before the semicolon or curly bracket (if any).
- Assume an open/left curly bracket { immediately after any method header or class header that does not already have one.
- Assume an appropriate amount of closing brackets before any method header to close all open brackets from the previous method or constructor.
- If there are missing curly brackets, clear indentation can "convey intent". Evaluate the response accordingly.
- Inside a method with left-justified code, indentation cannot "convey intent", so missing curly brackets cannot be assumed. Without bracketing or indentation, only the first line of a `while` / `if` / `for` is controlled by the statement; with open curly bracket and no indentation cues, the entire remainder of the method is "inside" the statement.

No Penalty

- `:` instead of `;` and vice versa
- `,` instead of `;` and vice versa

Question 3: Array / ArrayList**9 points****Canonical solution**

a.

```
public Round(String[] names)
{
    competitorList = new ArrayList<Competitor>();
    for(int i = 0; i < names.length; i++)
    {
        Competitor c = new Competitor(names[i], i + 1);
        competitorList.add(c);
    }
}
```

4 points

b.

```
public ArrayList<Match> buildMatches()
{
    ArrayList<Match> matches = new ArrayList<Match>();

    if (competitorList.size() % 2 == 0)
    {
        for(int i = 0; i < competitorList.size()/2; i++)
        {
            matches.add(new Match(competitorList.get(i),
                                   competitorList.get(competitorList.size() - i - 1)));
        }
    }
    else
    {
        for(int i = 1; i < competitorList.size() / 2 + 1; i++)
        {
            matches.add(new Match(competitorList.get(i),
                                   competitorList.get(competitorList.size() - i)));
        }
    }
    return matches;
}
```

5 points

a. Round

Scoring Criteria		Decision Rules	
1	Accesses* all elements of <code>names</code> (<i>no bounds errors</i>)	Responses will not earn the point if they • <code>access names</code> incorrectly	1 point
2	Initializes and maintains rank associated with each accessed competitor, beginning at 1	Responses can still earn the point even if they • maintain a rank variable with initial value 0, as long as it is offset by 1 when accessed • fail to construct a <code>Competitor</code> object Responses will not earn the point if they • compute rank incorrectly for any competitor	1 point
3	Constructs <code>Competitor</code> with provided name and computed rank	Responses can still earn the point even if they • access names incorrectly • compute rank incorrectly Responses will not earn the point if they • omit <code>new</code> in the construction of a <code>Competitor</code> or fail to use the correct number and type of parameters	1 point
4	Adds all constructed competitors to <code>competitorList</code> in the correct order (<i>algorithm</i>)	Responses can still earn the point even if they • fail to initialize <code>competitorList</code> (<i>ArrayList creation not assessed in this part</i>) • omit <code>new</code> in the construction of a <code>Competitor</code> • fail to access all elements of <code>names</code> • add elements with an incorrect or missing rank Responses will not earn the point if they • add items without constructing a <code>Competitor</code> • access <code>competitorList</code> incorrectly • fail to update the instance variable <code>competitorList</code> (e.g. by redeclaring as a local variable) • print or return a value instead of or in addition to updating <code>competitorList</code>	1 point

*An enhanced `for` loop inherently accesses all elements of an `ArrayList`

b. buildMatches

	Scoring Criteria	Decision Rules	
5	Declares and initializes local <code>ArrayList</code> of <code>Match</code> objects	Responses will not earn the point if they fail to declare the <code>ArrayList</code>, even if they have other local variables declared	1 point
6	Initializes and maintains an index used to move from both ends of <code>competitorList</code> (<i>algorithm</i>)	Responses can still earn the point even if they - start at 0 instead of 1 or vice versa - make a bounds error with the "high" index, as long as it is counting down from the end of the list - maintain two separate index variables instead of a single offset or other equivalent strategy, as long as it is used to count up from the start and down from the end - fail to use the indices to construct a <code>Match</code>	1 point
7	Computes starting low index based on even/odd comparison	Responses can still earn the point even if they - compute the index incorrectly, as long as the even/odd cases start at different indices - call the <code>size</code> method incorrectly - modifies <code>competitorList</code> instead of skipping an index Responses will not earn the point if they - determine even and odd number of competitors incorrectly	1 point
8	Gets two elements of <code>competitorList</code> based on maintained index, constructs a <code>Match</code> object from them, and adds it to the local list	Responses can still earn the point even if they - omit <code>new</code> in the creation of the <code>Match</code> object (<i>use of <code>new</code> not assessed for this type</i>) - use incorrect indices Responses will not earn the point if they - access <code>competitorList</code> incorrectly - fail to create an object of type <code>Match</code> - use incorrect number or type of parameters on any of the method calls	1 point

9	Populates the list with the correct number of pairs of competitors, omitting the first competitor in the odd case, without bounds errors (<i>algorithm</i>)	Responses can still earn the point even if they - fail to return the <code>ArrayList</code> (<i>return not assessed in this question</i>) - determine even and odd number of competitors incorrectly, as long as the cases are unambiguous - fail to create or use a <code>Match</code> object	1 point
		Responses will not earn the point if they - fail to guard against even and odd numbers of competitors - include the first competitor for odd numbers of competitors or omit the first for even numbers - include too many matches (e.g. due to continuing past the middle of the list) - add indices instead of competitors - make syntactically incorrect call(s) to <code>size</code> or other <code>ArrayList</code> methods - permanently modify <code>competitorList</code>	
	Question-specific penalties		
	None		

Alternate canonical solution:

```
public Round(String[] names)
{
    competitorList = new ArrayList<Competitor>();

    int rank = 1;

    for (String name : names)
    {
        Competitor c = new Competitor(name, rank);
        competitorList.add(c);
        rank++;
    }
}

public ArrayList<Match> buildMatches()
{
    ArrayList<Match> matches = new ArrayList<Match>();

    int low = 0;
    if (competitorList.size() % 2 == 1)
    {
        low = 1;
    }
    int high = competitorList.size() - 1;

    while (low < high)
    {
        Match m = new Match(competitorList.get(low),
                             competitorList.get(high));
        matches.add(m);
        low++;
        high--;
    }
    return matches;
}
```

Applying the Scoring Criteria

Apply the question scoring criteria 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

- Extraneous code with no side-effect (e.g., valid 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 ($\times$ • $\div$ $\leq$ $\geq$ $<>$ $\neq$)
- `[]` 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, "ArrayList" 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.*

2025 Digital Decision Rules

- Some non-ASCII characters are not printing correctly in ONE, particularly extended punctuation. Certain kinds of double-quotes may display as â[?] [?]; a character that displays as ï¼[?] may be a parenthesis, comma, or semicolon (or other punctuation). If the badly displayed character makes sense as one of those, evaluate the response accordingly.
- If there are missing closed-double-quotes or closed-parentheses, assume they are at the end of the line where they opened, immediately before the semicolon or curly bracket (if any).
- Assume an open/left curly bracket { immediately after any method header or class header that does not already have one.
- Assume an appropriate amount of closing brackets before any method header to close all open brackets from the previous method or constructor.
- If there are missing curly brackets, clear indentation can "convey intent". Evaluate the response accordingly.
- Inside a method with left-justified code, indentation cannot "convey intent", so missing curly brackets cannot be assumed. Without bracketing or indentation, only the first line of a `while` / `if` / `for` is controlled by the statement; with open curly bracket and no indentation cues, the entire remainder of the method is "inside" the statement.

No Penalty

- `:` instead of `;` and vice versa
- `,` instead of `;` and vice versa

2019

AP[®]

CollegeBoard

AP[®] Computer Science A

Scoring Guidelines

2019 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., 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. `<>`
- o `=` 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.*

2019 SCORING GUIDELINES**Question 1: Calendar**

Part (a)	<code>numberOfLeapYears</code>	5 points
-----------------	--------------------------------	-----------------

Intent: *Return the number of leap years in a range*

- +1** Initializes a numeric variable
- +1** Loops through each necessary year in the range
- +1** Calls `isLeapYear` on some valid year in the range
- +1** Updates count based on result of calling `isLeapYear`
- +1** Returns count of leap years

Part (b)	<code>dayOfWeek</code>	4 points
-----------------	------------------------	-----------------

Intent: *Return an integer representing the day of the week for a given date*

- +1** Calls `firstDayOfYear`
- +1** Calls `dayOfYear`
- +1** Calculates the value representing the day of the week
- +1** Returns the calculated value

Question-Specific Penalties

- 1** (t) Static methods called with `this`.

2019 SCORING GUIDELINES**Question 1: Scoring Notes**

Part (a) <code>numberOfLeapYears</code>			5 points
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Initializes a numeric variable		use the variable for loop control only
+1	Loops through each necessary year in the range		consider years outside the range
+1	Calls <code>isLeapYear</code> on some valid year in the range	do not use a loop	
+1	Updates count based on result of calling <code>isLeapYear</code>	- do not use a loop - do not initialize the counter	use result as a non-boolean
+1	Returns count of leap years	- loop from <code>year1</code> to <code>year2</code> incorrectly - do not initialize the counter	- do not use a loop - update or initialize the counter incorrectly - return early inside the loop
Part (b) <code>dayOfWeek</code>			4 points
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Calls <code>firstDayOfYear</code>		do not use the given year
+1	Calls <code>dayOfYear</code>		have arguments out of order
+1	Calculates the value representing the day of the week		make any error in the calculation
+1	Returns the calculated value	return the value from calling <code>firstDayOfYear</code> or <code>dayOfYear</code>	return a constant value

2019 SCORING GUIDELINES**Question 1: Calendar***Part (a)*

```
public static int numberOfLeapYears(int year1, int year2)
{
    int count = 0;
    for (int y = year1; y <= year2; y++)
    {
        if (isLeapYear(y))
        {
            count++;
        }
    }
    return count;
}
```

Part (b)

```
public static int dayOfWeek(int month, int day, int year)
{
    int startDay = firstDayOfYear(year);
    int nthDay = dayOfYear(month, day, year);
    int returnDay = (startDay + nthDay - 1) % 7;
    return returnDay;
}
```

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.

2019 SCORING GUIDELINES**Question 2: Step Tracker****Class:** `StepTracker`**9 points****Intent:** *Define implementation of a class to record fitness data*

- +1** Declares all appropriate `private` instance variables
- +2** Constructor
 - +1** Declares header: `public StepTracker(int ____)`
 - +1** Uses parameter and appropriate values to initialize instance variables
- +3** `addDailySteps` method
 - +1** Declares header: `public void addDailySteps(int ____)`
 - +1** Identifies active days and increments count
 - +1** Updates other instance variables appropriately
- +1** `activeDays` method
 - +1** Declares and implements `public int activeDays()`
- +2** `averageSteps` method
 - +1** Declares header: `public double averageSteps()`
 - +1** Returns calculated `double` average number of steps

2019 SCORING GUIDELINES**Question 2: Scoring Notes**

Class StepTracker			9 points
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Declares all appropriate <code>private</code> instance variables		- omit keyword <code>private</code> - declare variables outside the class
+2	Constructor		
+1	Declares header: <code>public StepTracker(int ____)</code>	omit keyword <code>public</code>	declare method <code>private</code>
+1	Uses parameter and appropriate values to initialize instance variables	initialize primitive instance variables to default values when declared	- fail to use the parameter to initialize some instance variable - fail to declare instance variables - initialize local variables instead of instance variables - assign variables to parameters
+3	addDailySteps method		
+1	Declares header: <code>public void addDailySteps(int ____)</code>	omit keyword <code>public</code>	declare method <code>private</code>
+1	Identifies active days and increments count	put valid comparison erroneously in some other method	- fail to use the parameter as part of the comparison - fail to increment a count of active days - fail to increment an instance variable - compare parameter to some numeric constant
+1	Updates other instance variables appropriately		- update another instance variable only on active days - update another instance variable inappropriately - fail to update appropriate instance variable - update a local variable
+1	activeDays method		
+1	Declares and implements <code>public int activeDays()</code>	return appropriate count of active days where the instance variables were updated improperly in <code>addDailySteps</code> or <code>activeDays</code>	- declare method <code>private</code> - return value that is not the number of active days - fail to return a value

2019 SCORING GUIDELINES**Question 2: Scoring Notes (continued)**

Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+2	<code>averageSteps</code> method		
+1	Declares header: <code>public double averageSteps()</code>	omit keyword <code>public</code>	declare method <code>private</code>
+1	Returns calculated <code>double average</code> number of steps	- maintain instance variables improperly but calculate appropriate average - fail to handle the special case where no days are tracked	- use integer division - calculate something other than steps divided by days - fail to return

2019 SCORING GUIDELINES**Question 2: Step Tracker**

```
public class StepTracker
{
    private int minSteps;
    private int totalSteps;
    private int numDays;
    private int numActiveDays;

    public StepTracker(int threshold)
    {
        minSteps = threshold;
        totalSteps = 0;
        numDays = 0;
        numActiveDays = 0;
    }

    public void addDailySteps(int steps)
    {
        totalSteps += steps;
        numDays++;
        if (steps >= minSteps)
        {
            numActiveDays++;
        }
    }

    public int activeDays()
    {
        return numActiveDays;
    }

    public double averageSteps()
    {
        if (numDays == 0)
        {
            return 0.0;
        }
        else
        {
            return (double) totalSteps / numDays;
        }
    }
}
```

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.

2019 SCORING GUIDELINES**Question 3: Delimiters**

Part (a)	<code>getDelimitersList</code>	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)	<code>isBalanced</code>	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

2019 SCORING GUIDELINES

Question 3: Scoring Notes

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

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.

2019 SCORING GUIDELINES**Question 4: Light Board**

Part (a)	<code>LightBoard</code>	4 points
-----------------	-------------------------	-----------------

Intent: *Define implementation of a constructor that initializes a 2D array of lights*

- +1** Creates a new `boolean[numRows][numCols]` and assigns to instance variable `lights`
- +1** Accesses all elements in the created 2D array (*no bounds errors*)
- +1** Computes the 40% probability
- +1** Sets all values of 2D array based on computed probability

Part (b)	<code>evaluateLight</code>	5 points
-----------------	----------------------------	-----------------

Intent: *Evaluate the status of a light in a 2D array of lights*

- +1** Accesses an element of `lights` as a `boolean` value in an expression
- +1** Traverses specified `col` of a 2D array (*no bounds errors*)
- +1** Counts the number of `true` values in the traversal
- +1** Performs an even calculation and a multiple of three calculation
- +1** Returns `true` or `false` according to all three rules

Question-Specific Penalties

- 1** (z) Constructor returns a value
- 1** (y) Destruction of persistent data

2019 SCORING GUIDELINES

Question 4: Scoring Notes

Part (a) <code>LightBoard</code>			4 points
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Creates a new <code>boolean[numRows][numCols]</code> and assigns to instance variable <code>lights</code>		- initialize a local variable that is never assigned to <code>lights</code> - omit the keyword <code>new</code> - use a type other than <code>boolean</code>
+1	Accesses all elements in the created 2D array (<i>no bounds errors</i>)	fail to create <code>lights</code> but assume <code>lights[numRows][numCols]</code>	
+1	Computes the 40% probability	use <code>Math.random() <= .4</code>	incorrectly cast to <code>int</code>
+1	Sets all values of 2D array based on computed probability	only assign <code>true</code> values	- compute a single probability but use it multiple times - reverse the sense of the comparison when assigning
Part (b) <code>evaluateLight</code>			5 points
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Accesses an element of <code>lights</code> as a <code>boolean</code> value in an expression		access <code>lights</code> as a type other than <code>boolean</code>
+1	Traverses specified <code>col</code> of a 2D array (<i>no bounds errors</i>)		
+1	Counts the number of <code>true</code> values in the traversal	- access too many or too few items in a single column - access a single row instead of a single column	count an item more than once
+1	Performs an even calculation and a multiple of three calculation		use <code>/</code> instead of <code>%</code>
+1	Returns <code>true</code> or <code>false</code> according to all three rules	have an incorrect column count but use the correct logic	- fail to return a value in some case - implement counting loop more than once with one loop that is incorrect

2019 SCORING GUIDELINES**Question 4: Light Board***Part (a)*

```
public LightBoard(int numRows, int numCols)
{
    lights = new boolean[numRows][numCols];

    for (int r = 0; r < numRows; r++)
    {
        for (int c = 0; c < numCols; c++)
        {
            double rnd = Math.random();
            lights[r][c] = rnd < 0.4;
        }
    }
}
```

Part (b)

```
public boolean evaluateLight(int row, int col)
{
    int numOn = 0;

    for (int r = 0; r < lights.length; r++)
    {
        if (lights[r][col])
        {
            numOn++;
        }
    }

    if (lights[row][col] && numOn % 2 == 0)
    {
        return false;
    }
    if (!lights[row][col] && numOn % 3 == 0)
    {
        return true;
    }
    return lights[row][col];
}
```

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.