

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

(a)

```
public double getAverageRating()
{
    int sum = 0;

    for (Review r : allReviews)
    {
        sum += r.getRating();
    }

    return (double) sum / allReviews.length;
}
```

3 points

(b)

```
public ArrayList<String> collectComments()
{
    ArrayList<String> commentList = new ArrayList<String>();

    for (int i = 0; i < allReviews.length; i++)
    {
        String comment = allReviews[i].getComment();
        if (comment.indexOf("!") >= 0)
        {
            String last =
                comment.substring(comment.length() - 1);
            if (!last.equals("!") && !last.equals("."))
            {
                comment += ".";
            }
            commentList.add(i + "-" + comment);
        }
    }
    return commentList;
}
```

6 points

(a) `getAverageRating`

Scoring Criteria		Decision Rules	
1	Initializes and accumulates sum	Response can still earn the point even if they - fail to use a loop to accumulate - fail to call <code>getRating</code> or call <code>getRating</code> incorrectly	1 point
2	Accesses every element of <code>allReviews</code> (<i>no bounds errors</i>)	Responses will not earn the point if they access the elements of <code>allReviews</code> incorrectly	1 point
3	Computes and returns <code>double</code> average rating based on <code>getRating</code> return values (<i>algorithm</i>)	Response can still earn the point even if they - fail to initialize the accumulator for the sum Responses will not earn the point if they - fail to accumulate the sum of all ratings - use integer division to compute average - include parameters on call to <code>getRating</code> - fail to call <code>getRating</code> on all elements of <code>allReviews</code>	1 point
Total for part (a)			3 points

(b) `collectComments`

Scoring Criteria		Decision Rules	
4	Instantiates an <code>ArrayList</code> capable of holding <code>String</code> objects		1 point
5	Accesses every element of <code>allReviews</code> (<i>no bounds errors</i>)	Responses can still earn the point even if they - fail to keep track of the index Responses will not earn the point if they - access the elements of <code>allReviews</code> incorrectly	1 point
6	Calls <code>getComment</code> on an element of <code>allReviews</code> , calls at least one <code>String</code> method appropriately on the <code>getComment</code> return value, and all <code>String</code> method calls are syntactically	Responses can still earn the point even if they - call some of the <code>String</code> methods on objects other than <code>getComment</code> return values Responses will not earn the point if they - include a parameter when calling <code>getComment</code> - call any <code>String</code> methods incorrectly - call any <code>String</code> methods on objects other than <code>String</code> values	1 point
7	Compares the final character of the comment to both a period and an exclamation point	Responses can still earn the point even if they - use incorrect logic in the comparison - call <code>String</code> methods incorrectly Responses will not earn the point if they - use <code>==</code> instead of <code>equals</code> when comparing <code>String</code> objects	1 point
8	Assembles string appropriately based on result of comparison of last character with period and exclamation point (<i>algorithm</i>)	Responses can still earn the point even if they - call <code>String</code> methods incorrectly - use <code>==</code> instead of <code>equals</code> Responses will not earn the point if they - fail to keep track of the element index - use incorrect logic in the comparison	1 point
9	Adds all and only appropriate constructed strings to the <code>ArrayList</code> (<i>algorithm</i>)	Responses can still earn the point even if they - initialize the <code>ArrayList</code> incorrectly - fail to return the constructed <code>ArrayList</code> (<i>return is not assessed</i>) - assemble the review string incorrectly - access the elements of <code>allReviews</code> incorrectly	1 point

	Total for part (b)	6 points
Question-specific penalties		
None		
	Total for question 3	9 points

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 (`x • ÷ ≤ ≥ <> ≠`)
- `[]` 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.*

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> (no bounds errors)	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 (must be in the context of a loop)	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> (no bounds errors)	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.

2018**AP[®]** **CollegeBoard**

AP Computer Science A

Scoring Guidelines

2018 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, “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.

2018 SCORING GUIDELINES

Question 1: Frog Simulation

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

Intent: *Simulate the distance traveled by a hopping frog*

- +1** Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position
- +1** Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)
- +1** Determines if a distance representing multiple hops is at least `goalDistance`
- +1** Determines if a distance representing multiple hops is less than starting position
- +1** Returns `true` if goal ever reached, `false` if goal never reached or position ever less than starting position

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

Intent: *Determine the proportion of successful frog hopping simulations*

- +1** Calls `simulate` the specified number of times (*no bounds errors*)
- +1** Initializes and accumulates a count of `true` results
- +1** Calculates proportion of successful simulations using `double` arithmetic
- +1** Returns calculated value

2018 SCORING GUIDELINES

Question 1: Scoring Notes

Part (a) <code>simulate</code>			5 points
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls <code>hopDistance</code> and uses returned distance to adjust (or represent) the frog's position	use <code>hopDistance()</code> as a position, like <code>hopDistance() < 0</code>	only use <code>hopDistance()</code> as a count, like <code>hopDistance() < maxHops</code>
+1	Initializes and accumulates the frog's position at most <code>maxHops</code> times (<i>must be in context of a loop</i>)		do not use a loop
+1	Determines if a distance representing multiple hops is at least <code>goalDistance</code>	use some number of hops * <code>hopDistance()</code> as the frog's final position	
+1	Determines if a distance representing multiple hops is less than starting position		
+1	Returns <code>true</code> if goal ever reached, <code>false</code> if goal never reached or position ever less than starting position	have checks for all three conditions and correct return logic based on those checks, even if a check did not earn a point	- do not check all three conditions - only check for <code>goalDistance</code> after the loop - only check for starting position after the loop
Part (b) <code>runSimulations</code>			4 points
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls <code>simulate</code> the specified number of times (<i>no bounds errors</i>)	do not use the result of calling <code>simulate</code>	do not use a loop
+1	Initializes and accumulates a count of <code>true</code> results		- initialize the count inside a loop - do not use a loop
+1	Calculates proportion of successful simulations using <code>double</code> arithmetic	perform the correct calculation on an accumulated value, even if there was an error in the accumulation	fail to divide by the parameter
+1	Returns calculated value		- calculate values using nonnumeric types - return a count of simulations

2018 SCORING GUIDELINES**Question 1: Frog Simulation***Part (a)*

```
public boolean simulate()
{
    int position = 0;

    for (int count = 0; count < maxHops; count++)
    {
        position += hopDistance();
        if (position >= goalDistance)
        {
            return true;
        }
        else if (position < 0)
        {
            return false;
        }
    }
    return false;
}
```

Part (b)

```
public double runSimulations(int num)
{
    int countSuccess = 0;

    for (int count = 0; count < num; count++)
    {
        if(simulate())
        {
            countSuccess++;
        }
    }
    return (double)countSuccess / num;
}
```

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.

2018 SCORING GUIDELINES

Question 2: Word Pair

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

Intent: *Form pairs of strings from an array and add to an `ArrayList`*

- +1** Creates new `ArrayList` and assigns to `allPairs`
- +1** Accesses all elements of `words` (*no bounds errors*)
- +1** Constructs new `WordPair` using distinct elements of `words`
- +1** Adds all necessary pairs of elements from word array to `allPairs`
- +1** **On exit:** `allPairs` contains all necessary pairs and no unnecessary pairs

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

Intent: *Count the number of pairs in an `ArrayList` that have the same value*

- +1** Accesses all elements in `allPairs` (*no bounds errors*)
- +1** Calls `getFirst` or `getSecond` on an element from list of pairs
- +1** Compares first and second components of a pair in the list
- +1** Counts number of matches of pair-like values

Question-Specific Penalties

- 1** (z) Constructor returns a value

2018 SCORING GUIDELINES

Question 2: Scoring Notes

Part (a) <code>WordPairList</code>			5 points
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Creates new <code>ArrayList</code> and assigns to <code>allPairs</code>	<code>allPairs = new ArrayList();</code> <code>allPairs = new ArrayList<>();</code> <code>this.allPairs = ...</code>	initialize a local variable that is never assigned to <code>allPairs</code>
+1	Accesses all elements of <code>words</code> (<i>no bounds errors</i>)		
+1	Constructs new <code>WordPair</code> using distinct elements of <code>words</code>		
+1	Adds all necessary pairs of elements from word array to <code>allPairs</code>	- have a loop bounds error - add unnecessary pairs	- improperly add to an <code>ArrayList</code>, e.g., <code>allPairs.get(i) = x;</code> - only add consecutive pairs (<code>words[i], words[i+1]</code>)
+1	On exit: <code>allPairs</code> contains all necessary pairs and no unnecessary pairs	- improperly add to an <code>ArrayList</code>, e.g., <code>allPairs.get(i) = x;</code> - have a loop bounds error	add pairs <code>(i, i)</code> or <code>(i, j)</code> where <code>i > j</code>
Part (b) <code>numMatches</code>			4 points
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Accesses all elements in <code>allPairs</code> (<i>no bounds errors</i>)		access elements of <code>allPairs</code> as array elements (e.g., <code>allPairs[i]</code>)
+1	Calls <code>getFirst</code> or <code>getSecond</code> on an element from list of pairs		
+1	Compares first and second components of a pair in the list		compare using <code>==</code>
+1	Counts number of matches of pair-like values		fail to initialize the counter

Return is not assessed in part (b).

2018 SCORING GUIDELINES**Question 2: Word Pair***Part (a)*

```
public WordPairList(String[] words)
{
    allPairs = new ArrayList<WordPair>();

    for (int i = 0; i < words.length-1; i++)
    {
        for (int j = i+1; j < words.length; j++)
        {
            allPairs.add(new WordPair(words[i], words[j]));
        }
    }
}
```

Part (b)

```
public int numMatches()
{
    int count = 0;

    for (WordPair pair: allPairs)
    {
        if (pair.getFirst().equals(pair.getSecond()))
        {
            count++;
        }
    }
    return count;
}
```

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.

2018 SCORING GUIDELINES

Question 3: Code Word Checker

Class: `CodeWordChecker`

9 points

Intent: *Define implementation of a class to determine if a string meets a set of criteria*

- +1** Declares header: `public class CodeWordChecker implements StringChecker`
- +1** Declares all appropriate `private` instance variables
- +3** Constructors
 - +1** Declares headers: `public CodeWordChecker(int __, int __, String __)` and `public CodeWordChecker(String __)`
 - +1** Uses all parameters to initialize instance variables in 3-parameter constructor
 - +1** Uses parameter and default values to initialize instance variables in 1-parameter constructor
- +4** `isValid` method
 - +1** Declares header: `public boolean isValid(String __)`
 - +1** Checks for length between min and max inclusive
 - +1** Checks for unwanted string
 - +1** Returns `true` if length is between min and max and does not contain the unwanted string, `false` otherwise

2018 SCORING GUIDELINES

Question 3: Scoring Notes

Class <code>CodeWordChecker</code>		9 points	
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Declares header: <code>public class</code> <code>CodeWordChecker</code> <code>implements</code> <code>StringChecker</code>	omit keyword <code>public</code>	- declare class <code>private</code> - declare class <code>static</code>
+1	Declares all appropriate <code>private</code> instance variables		- declare variables as <code>static</code> - omit keyword <code>private</code> - declare variables outside the class
+3	Constructors		
+1	Declares headers: <code>public</code> <code>CodeWordChecker</code> <code>(int __, int __,</code> <code>String __)</code> and <code>public</code> <code>CodeWordChecker</code> <code>(String __)</code>	omit keyword <code>public</code>	- declare method <code>static</code> - declare method <code>private</code>
+1	Uses all parameters to initialize instance variables in 3- parameter constructor		- fail to declare instance variables - initialize local variables instead of instance variables - assign variables to parameters
+1	Uses parameter and default values to initialize instance variables in 1- parameter constructor	initialize instance variables to default values when declared	- fail to declare instance variables - initialize local variables instead of instance variables - assign variables to parameters
+4	<code>isValid</code> method		
+1	Declares header: <code>public boolean</code> <code>isValid</code> <code>(String __)</code>		- fail to declare method <code>public</code> - declare method <code>static</code>
+1	Checks for length between min and max inclusive		- fail to use instance variables - fail to declare the method header
+1	Checks for unwanted string		- fail to use instance variables - fail to declare the method header
+1	Returns <code>true</code> if length is between min and max and does not contain the unwanted string, <code>false</code> otherwise	have incorrect checks for length and/or containment, but return the correct value based on those checks	- fail to declare the method header - fail to return in all cases - only check one substring location for containment

2018 SCORING GUIDELINES**Question 3: Code Word Checker**

```
public class CodeWordChecker implements StringChecker
{
    private int minLength;
    private int maxLength;
    private String notAllowed;

    public CodeWordChecker(int minLen, int maxLen, String symbol)
    {
        minLength = minLen;
        maxLength = maxLen;
        notAllowed = symbol;
    }

    public CodeWordChecker(String symbol)
    {
        minLength = 6;
        maxLength = 20;
        notAllowed = symbol;
    }

    public boolean isValid(String str)
    {
        return str.length() >= minLength && str.length() <= maxLength &&
            str.indexOf(notAllowed) == -1;
    }
}
```

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.

2018 SCORING GUIDELINES

Question 4: Latin Squares

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

Intent: Create a 1-D array that contains the values from one column of a 2-D array

- +1** Constructs a new `int` array of size `arr2D.length`
- +1** Accesses all items in one column of `arr2D` (*no bounds errors*)
- +1** Assigns one element from `arr2D` to the corresponding element in the new array
- +1 On exit:** The new array has all the elements from the specified column in `arr2D` in the correct order

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

Intent: Check conditions to determine if a square 2-D array is a Latin square

- +1** Calls `containsDuplicates` referencing a row or column of `square`
- +1** Calls `hasAllValues` referencing two different rows, two different columns, or one row and one column
- +1** Applies `hasAllValues` to all rows or all columns (*no bounds errors*)
- +1** Calls `getColumn` to obtain a valid column from `square`
- +1** Returns `true` if all three Latin square conditions are satisfied, `false` otherwise

Question-Specific Penalties

- 1** (r) incorrect construction of a copy of a row
- 1** (s) syntactically incorrect method call to any of `getColumn()`, `containsDuplicates()`, or `hasAllValues()`

2018 SCORING GUIDELINES

Question 4: Scoring Notes

Part (a) <code>getColumn</code>			4 points
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Constructs a new <code>int</code> array of size <code>arr2D.length</code>		only create an <code>ArrayList</code>
+1	Accesses all items in one column of <code>arr2D</code> (<i>no bounds errors</i>)	declare the new array of an incorrect size and use that size as the number of loop iterations	switch row and column indices
+1	Assigns one element from <code>arr2D</code> to the corresponding element in the new array		use <code>ArrayList</code> methods to add to array
+1	On exit: The new array has all the elements from the specified column in <code>arr2D</code> in the correct order		- switch row and column indices - do not use an index when assigning values to the array
Part (b) <code>isLatin</code>			5 points
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls <code>containsDuplicates</code> referencing a row or column of <code>square</code>	reference any row or column of <code>square</code>, even if the syntax of the reference is incorrect	
+1	Calls <code>hasAllValues</code> referencing two different rows, two different columns, or one row and one column	reference any two distinct rows, two distinct columns, or a row and column of <code>square</code>, even if the syntax of the reference is incorrect	
+1	Applies <code>hasAllValues</code> to all rows or all columns (<i>no bounds errors</i>)		only reference one array in the call to <code>hasAllValues</code>
+1	Calls <code>getColumn</code> to obtain a valid column from <code>square</code>		reverse parameters
+1	Returns <code>true</code> if all three Latin square conditions are satisfied, <code>false</code> otherwise	test the three sets of conditions and return the correct value	

Return is not assessed in Part (a).

2018 SCORING GUIDELINES**Question 4: Latin Squares***Part (a)*

```
public static int[] getColumn(int[][] arr2D, int c)
{
    int[] result = new int[arr2D.length];

    for (int r = 0; r < arr2D.length; r++)
    {
        result[r] = arr2D[r][c];
    }
    return result;
}
```

Part (b)

```
public static boolean isLatin(int[][] square)
{
    if (containsDuplicates(square[0]))
    {
        return false;
    }

    for (int r = 1; r < square.length; r++)
    {
        if (!hasAllValues(square[0], square[r]))
        {
            return false;
        }
    }

    for (int c = 0; c < square[0].length; c++)
    {
        if (!hasAllValues(square[0], getColumn(square, c)))
        {
            return false;
        }
    }

    return true;
}
```

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.