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

- 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 (× • ÷ < > ≠)
- [] 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, “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.
Question 1: Frog Simulation

**Part (a)**

| simulate | 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)**

| runSimulations | 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
## Question 1: Scoring Notes

### Part (a) `simulate`

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point if they...</th>
<th>Responses will not earn the point if they...</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Calls <code>hopDistance</code> and uses returned distance to adjust (or represent) the frog’s position</td>
<td>• use <code>hopDistance()</code> as a position, like <code>hopDistance() &lt; 0</code></td>
<td>• only use <code>hopDistance()</code> as a count, like <code>hopDistance() &lt; maxHops</code></td>
</tr>
<tr>
<td>+1</td>
<td>Initializes and accumulates the frog’s position at most <code>maxHops</code> times (<em>must be in context of a loop</em>)</td>
<td></td>
<td>• do not use a loop</td>
</tr>
<tr>
<td>+1</td>
<td>Determines if a distance representing multiple hops is at least <code>goalDistance</code></td>
<td>• use some number of hops * <code>hopDistance()</code> as the frog’s final position</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Determines if a distance representing multiple hops is less than starting position</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Returns true if goal ever reached, false if goal never reached or position ever less than starting position</td>
<td>• have checks for all three conditions and correct return logic based on those checks, even if a check did not earn a point</td>
<td>• do not check all three conditions • only check for <code>goalDistance</code> after the loop • only check for starting position after the loop</td>
</tr>
</tbody>
</table>

### Part (b) `runSimulations`

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point if they...</th>
<th>Responses will not earn the point if they...</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Calls <code>simulate</code> the specified number of times (<em>no bounds errors</em>)</td>
<td>• do not use the result of calling <code>simulate</code></td>
<td>• do not use a loop</td>
</tr>
<tr>
<td>+1</td>
<td>Initializes and accumulates a count of true results</td>
<td></td>
<td>• initialize the count inside a loop • do not use a loop</td>
</tr>
<tr>
<td>+1</td>
<td>Calculates proportion of successful simulations using double arithmetic</td>
<td>• perform the correct calculation on an accumulated value, even if there was an error in the accumulation</td>
<td>• fail to divide by the parameter</td>
</tr>
<tr>
<td>+1</td>
<td>Returns calculated value</td>
<td></td>
<td>• calculate values using nonnumeric types • return a count of simulations</td>
</tr>
</tbody>
</table>
Question 1: Frog Simulation

Part (a)

```java
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)

```java
public double runSimulations(int num)
{
    int countSuccess = 0;
    for (int count = 0; count < num; count++)
    {
        if(simulate())
        {
            countSuccess++;
        }
    }
    return (double)countSuccess / num;
}
```
Question 2: Word Pair

Part (a)

<table>
<thead>
<tr>
<th>Component</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordPairList</td>
<td>5 points</td>
</tr>
</tbody>
</table>

**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)

<table>
<thead>
<tr>
<th>Component</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>numMatches</td>
<td>4 points</td>
</tr>
</tbody>
</table>

**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 (2) Constructor returns a value
Question 2: Scoring Notes

### Part (a) WordPairList

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point if they...</th>
<th>Responses will not earn the point if they...</th>
</tr>
</thead>
</table>
| +1     | Creates new ArrayList and assigns to allPairs | allPairs = new ArrayList();  
allPairs = new ArrayList<>();  
this.allPairs = ... | initialize a local variable that is never assigned 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 | have a loop bounds error  
add unnecessary pairs | improperly add to an ArrayList, e.g., allPairs.get(i) = x;  
only add consecutive pairs (words[i], words[i+1]) |
| +1     | On exit: allPairs contains all necessary pairs and no unnecessary pairs | improperly add to an ArrayList, e.g., allPairs.get(i) = x;  
have a loop bounds error | add pairs (i, i) or (i, j) where i > j |

### Part (b) numMatches

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point if they...</th>
<th>Responses will not earn the point if they...</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Accesses all elements in allPairs (no bounds errors)</td>
<td></td>
<td>access elements of allPairs as array elements (e.g., allPairs[i])</td>
</tr>
<tr>
<td>+1</td>
<td>Calls getFirst or getSecond on an element from list of pairs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Compares first and second components of a pair in the list</td>
<td>compare using ==</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Counts number of matches of pair-like values</td>
<td>fail to initialize the counter</td>
<td></td>
</tr>
</tbody>
</table>

Return is not assessed in part (b).
Question 2: Word Pair

Part (a)

```java
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)

```java
public int numMatches() {
    int count = 0;
    for (WordPair pair: allPairs) {
        if (pair.getFirst().equals(pair.getSecond())) {
            count++;
        }
    }
    return count;
}
```
## 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
## Question 3: Scoring Notes

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

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.
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;
    }
}
Question 4: Latin Squares

Part (a)  getColumn  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)  isLatin  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()`
## Question 4: Scoring Notes

### Part (a) getColumn

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point if they...</th>
<th>Responses will not earn the point if they...</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Constructs a new int array of size <code>arr2D.length</code></td>
<td></td>
<td>• only create an <code>ArrayList</code></td>
</tr>
<tr>
<td>+1</td>
<td>Accesses all items in one column of <code>arr2D</code> <em>(no bounds errors)</em></td>
<td>• declare the new array of an incorrect size and use that size as the number of loop iterations</td>
<td>• switch row and column indices</td>
</tr>
<tr>
<td>+1</td>
<td>Assigns one element from <code>arr2D</code> to the corresponding element in the new array</td>
<td></td>
<td>• use <code>ArrayList</code> methods to add to array</td>
</tr>
<tr>
<td>+1</td>
<td><strong>On exit:</strong> The new array has all the elements from the specified column in <code>arr2D</code> in the correct order</td>
<td></td>
<td>• switch row and column indices</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• do not use an index when assigning values to the array</td>
</tr>
</tbody>
</table>

### Part (b) isLatin

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point if they...</th>
<th>Responses will not earn the point if they...</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Calls <code>containsDuplicates</code> referencing a row or column of <code>square</code></td>
<td>• reference any row or column of <code>square</code>, even if the syntax of the reference is incorrect</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Calls <code>hasAllValues</code> referencing two different rows, two different columns, or one row and one column</td>
<td>• 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</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Applies <code>hasAllValues</code> to all rows or all columns <em>(no bounds errors)</em></td>
<td></td>
<td>• only reference one array in the call to <code>hasAllValues</code></td>
</tr>
<tr>
<td>+1</td>
<td>Calls <code>getColumn</code> to obtain a valid column from <code>square</code></td>
<td></td>
<td>• reverse parameters</td>
</tr>
<tr>
<td>+1</td>
<td>Returns <code>true</code> if all three Latin square conditions are satisfied, <code>false</code> otherwise</td>
<td></td>
<td>• test the three sets of conditions and return the correct value</td>
</tr>
</tbody>
</table>

Return is not assessed in Part (a).
Question 4: Latin Squares

Part (a)

```java
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)

```java
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.