

Question 1: Methods and Control Structures**9 points****Canonical solution****(a)**

```
public int scoreGuess(String guess)
{
    int count = 0;

    for (int i = 0; i <= secret.length() - guess.length(); i++)
    {
        if (secret.substring(i, i + guess.length()).equals(guess))
        {
            count++;
        }
    }

    return count * guess.length() * guess.length();
}
```

5 points**(b)**

```
public String findBetterGuess(String guess1, String guess2)
{
    if (scoreGuess(guess1) > scoreGuess(guess2))
    {
        return guess1;
    }
    if (scoreGuess(guess2) > scoreGuess(guess1))
    {
        return guess2;
    }
    if (guess1.compareTo(guess2) > 0)
    {
        return guess1;
    }
    return guess2;
}
```

4 points

(a) scoreGuess

Scoring Criteria		Decision Rules	
1	Compares <code>guess</code> to a substring of <code>secret</code>	Responses can still earn the point even if they only call <code>secret.indexOf(guess)</code> Responses will not earn the point if they use <code>==</code> instead of <code>equals</code>	1 point
2	Uses a substring of <code>secret</code> with correct length for comparison with <code>guess</code>	Responses can still earn the point even if they - only call <code>secret.indexOf(guess)</code> - use <code>==</code> instead of <code>equals</code>	1 point
3	Loops through all necessary substrings of <code>secret</code> (<i>no bounds errors</i>)	Responses will not earn the point if they skip overlapping occurrences	1 point
4	Counts number of identified occurrences of <code>guess</code> within <code>secret</code> (<i>in the context of a condition involving both <code>secret</code> and <code>guess</code></i>)	Responses can still earn the point even if they - initialize count incorrectly or not at all - identify occurrences incorrectly	1 point
5	Calculates and returns correct final score (<i>algorithm</i>)	Responses will not earn the point if they - initialize count incorrectly or not at all - fail to use a loop - fail to compare <code>guess</code> to multiple substrings of <code>secret</code> - count the same matching substring more than once - use a changed or incorrect <code>guess</code> length when computing the score	1 point
Total for part (a)			5 points

(b) `findBetterGuess`

Scoring Criteria		Decision Rules	
6	Calls <code>scoreGuess</code> to get scores for <code>guess1</code> and <code>guess2</code>	Responses will not earn the point if they - fail to include parameters in the method calls - call the method on an object or class other than <code>this</code>	1 point
7	Compares the scores	Responses will not earn the point if they - only compare using <code>==</code> or <code>!=</code> - fail to use the result of the comparison in a conditional statement	1 point
8	Determines which of <code>guess1</code> and <code>guess2</code> is alphabetically greater	Responses can still earn the point even if they reverse the comparison Responses will not earn the point if they - reimplement <code>compareTo</code> incorrectly - use result of <code>compareTo</code> as if <code>boolean</code>	1 point
9	Returns the identified <code>guess1</code> or <code>guess2</code> (<i>algorithm</i>)	Responses can still earn the point even if they - call <code>scoreGuess</code> incorrectly - compare strings incorrectly Responses will not earn the point if they - reverse a comparison - omit either comparison - fail to return a guess in some case	1 point
Total for part (b)			4 points
Question-specific penalties			
None			
Total for question 1			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 ($\times$ • $\div$ $\leq$ $\geq$ $<>$ $\neq$)
- `[]` 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, “`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.*

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.