

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.*

Question 1: Methods and Control Structures**9 points****Canonical solution****(a)** **5 points**

```
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();
}
```

(b) **4 points**

```
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;
}
```

(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.*

2017

AP[®]

CollegeBoard

AP Computer Science A

Scoring Guidelines

2017 GENERAL 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.*

2017 SCORING GUIDELINES

Question 1: Digits

Part (a)	Digits constructor	5 points
-----------------	--------------------	-----------------

Intent: *Initialize instance variable using passed parameter*

- +1 Constructs `digitList`
- +1 Identifies a digit in `num`
- +1 Adds at least one identified digit to a list
- +1 Adds all identified digits to a list (*must be in context of a loop*)
- +1 **On exit:** `digitList` contains all and only digits of `num` in the correct order

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

Intent: *Determine whether or not elements in `digitList` are in increasing order*

- +1 Compares at least one identified consecutive pair of `digitList` elements
- +1 Determines if a consecutive pair of `digitList` is out of order (*must be in context of a `digitList` traversal*)
- +1 Compares all necessary consecutive pairs of elements (*no bounds errors*)
- +1 Returns `true` iff all consecutive pairs of elements are in order; returns `false` otherwise

Question-Specific Penalties

- 2 (q) Uses confused identifier instead of `digitList`

2017 SCORING GUIDELINES

Question 1: Scoring Notes

Part (a) Digits constructor			5 points
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Constructs digitList		- initialize a local variable instead of digitList - create an ArrayList<int>
+1	Identifies a digit in num	identify one digit of num or a length one substring/character of the String representation of num	- treat num itself as a String - convert num to a String incorrectly
+1	Adds at least one identified digit to a list	call add for some ArrayList using the previously identified digit, even if that digit was identified incorrectly	add String or char to digitList without proper conversion to the correct type
+1	Adds all identified digits to a list (must be in the context of a loop)	call add for some ArrayList using previously identified digits, even if those digits were identified incorrectly	identify only 1 digit
+1	On exit: digitList contains all and only digits of num in the correct order	add to digitList even if it is not instantiated properly	- obtain a list with the digits in reverse order - omit one or more digits - add extra digits - mishandle edge case, e.g., 0 or 10 - make a bounds error processing the String representation of num
Part (b) isStrictlyIncreasing			4 points
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Compares at least one identified consecutive pair of digitList elements	- compare two consecutive Integers using compareTo - explicitly convert two consecutive Integers to ints and compare those with >=, <= etc. - use auto-unboxing to convert two consecutive Integers to ints and compare those with >=, <= etc.	- access digitList as an array or string - fail to call .get () - compare using !>
+1	Determines if a consecutive pair of digitList is out of order (must be in context of a digitList traversal)	determine the correct relationship between the two compared consecutive elements, even if the syntax of the comparison is incorrect	fail to consider the case where the two elements are equal for the false case
+1	Compares all necessary consecutive pairs of elements (no bounds errors)		return early
+1	Returns true iff all consecutive pairs of elements are in order; returns false otherwise	compare consecutive pairs for inequality, but fail to consider the case when two elements are equal	return prematurely via if (...) return false; else return true;

2017 SCORING GUIDELINES**Question 1: Digits***Part (a)*

```
public Digits(int num)
{
    digitList = new ArrayList<Integer>();

    if (num == 0)
    {
        digitList.add(new Integer(0));
    }

    while (num > 0)
    {
        digitList.add(0, new Integer(num % 10));
        num /= 10;
    }
}
```

Part (b)

```
public boolean isStrictlyIncreasing()
{
    for (int i = 0; i < digitList.size()-1; i++)
    {
        if (digitList.get(i).intValue() >= digitList.get(i+1).intValue())
        {
            return false;
        }
    }
    return true;
}
```

Note: The solutions shown above were written in compliance with the AP Java subset methods listed for `Integer` objects. Students were allowed to use the automatic "boxing" and "unboxing" of `Integer` objects in their solutions, which eliminates the need to use `"new Integer(...)"` in part (a) and `"intValue()"` in part (b).

2017 SCORING GUIDELINES

Question 2: MultiPractice

Class:	MultiPractice	9 points
---------------	---------------	-----------------

Intent: *Define implementation of class to produce multiplication practice problems*

- +1** Declares header: `public class MultiPractice implements StudyPractice`
- +1** Declares all necessary `private` instance variables
- +2** Constructor
 - +1** Declares header: `public MultiPractice(int __, int __)`
 - +1** Initializes all instance variables using parameters
- +3** `getProblem` method
 - +1** Declares header: `public String getProblem()`
 - +1** Builds string with current values of instance variables
 - +1** Returns constructed string
- +2** `nextProblem` method
 - +1** Declares header: `public void nextProblem()`
 - +1** Updates instance variable(s) to reflect incremented second number

2017 SCORING GUIDELINES

Question 2: Scoring Notes

Class MultPractice			9 points
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Declares header: public class MultPractice implements StudyPractice	omit keyword <code>public</code>	declare class <code>private</code>
+1	Declares all necessary private instance variables	declare the unchanging instance variable as <code>final</code>	- declare variables as <code>static</code> - omit keyword <code>private</code>
+2	Constructor		
+1	Declares header: public MultPractice (int ____, int ____)	omit keyword <code>public</code>	
+1	Initializes all instance variables using parameters		- fail to declare nonlocal variables - initialize local variables instead of instance variables - assign variables to parameters
+3	getProblem method		
+1	Declares header: public String getProblem()		fail to declare method <code>public</code>
+1	Builds string with current values of instance variables	- write appropriate code in a method other than <code>getProblem</code> - make capitalization or spacing errors	- fail to declare nonlocal variables - fail to use instance variables - miscast <code>(String) intVar</code> - call <code>intVar.toString()</code>
+1	Returns constructed string		return a literal string
+2	nextProblem method		
+1	Declares header: public void nextProblem()		fail to declare method <code>public</code>
+1	Updates instance variable(s) to reflect incremented second number		fail to declare non-local variables

2017 SCORING GUIDELINES**Question 2: MultiPractice**

```
public class MultiPractice implements StudyPractice
{
    private int first;
    private int second;

    public MultiPractice(int num1, int num2)
    {
        first = num1;
        second = num2;
    }

    public String getProblem()
    {
        return first + " TIMES " + second;
    }

    public void nextProblem()
    {
        second++;
    }
}
```


2017 SCORING GUIDELINES

Question 3: PhraseEditor

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

Intent: *Replace the n th occurrence of a given string with a given replacement*

- +1** Calls `findNthOccurrence` to find the index of the n th occurrence
- +1** Preserves `currentPhrase` only if n th occurrence does not exist
- +1** Identifies components of `currentPhrase` to retain (uses `substring` to extract before/after)
- +1** Creates replacement string using identified components and `repl`
- +1** Assigns replacement string to instance variable (`currentPhrase`)

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

Intent: *Return the index of the last occurrence of a given string*

- +1** Calls `findNthOccurrence` to find the index of the n th occurrence
- +1** Increments (or decrements) the value used as n when finding n th occurrence
- +1** Returns the index of the last occurrence, if it exists
- +1** Returns -1 only when no occurrences exist

Question-Specific Penalties

- 1** (q) Uses `currentPhrase.findNthOccurrence`
- 2** (r) Confused identifier instead of `currentPhrase`

2017 SCORING GUIDELINES

Question 3: Scoring Notes

Part (a) <code>replaceNthOccurrence</code>			5 points
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Calls <code>findNthOccurrence</code> to find the index of the <code>nth</code> occurrence	do not use the result of calling <code>findNthOccurrence</code>	
+1	Preserves <code>currentPhrase</code> only if <code>nth</code> occurrence does not exist		fail to use a conditional
+1	Identifies components of <code>currentPhrase</code> to retain (uses <code>substring</code> to extract before/after)	identify start and end of substring to be replaced	
+1	Creates replacement string using identified components and <code>repl</code>		create a replacement string that is out of order
+1	Assigns replacement string to instance variable (<code>currentPhrase</code>)		
Part (b) <code>findLastOccurrence</code>			4 points
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Calls <code>findNthOccurrence</code> to find the index of the <code>nth</code> occurrence	do not use the result of calling <code>findNthOccurrence</code>	<code>return currentPhrase.lastIndexOf(str);</code> - call <code>findNthOccurrence</code> with an integer parameter of 0
+1	Increments (or decrements) the value used as <code>n</code> when finding <code>nth</code> occurrence	<code>return currentPhrase.lastIndexOf(str);</code> - advance through <code>currentPhrase</code> searching for <code>nth</code> occurrence of <code>str</code>	
+1	Returns the index of the last occurrence, if it exists	<code>return currentPhrase.lastIndexOf(str);</code> - compute the correct value to be returned in all cases, but no return statement exists for any case	- shorten string being searched - always return in first iteration of the loop
+1	Returns -1 only when no occurrences exist	<code>return currentPhrase.lastIndexOf(str);</code>	- compute the correct value to be returned in all cases, but no return statement exists for any case - always return in first iteration of the loop

2017 SCORING GUIDELINES**Question 3: PhraseEditor***Part (a)*

```
public void replaceNthOccurrence(String str, int n, String repl)
{
    int loc = findNthOccurrence(str, n);

    if (loc != -1)
    {
        currentPhrase = currentPhrase.substring(0, loc) + repl +
                        currentPhrase.substring(loc + str.length());
    }
}
```

Part (b)

```
public int findLastOccurrence(String str)
{
    int n = 1;
    while (findNthOccurrence(str, n+1) != -1)
    {
        n++;
    }
    return findNthOccurrence(str, n);
}
```

2017 SCORING GUIDELINES

Question 4: Successor Array

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

Intent: *Find the position of a given integer in a 2D integer array*

- +1** Accesses all necessary elements of `intArr` (*no bounds errors*)
- +1** Identifies `intArr` element equal to `num` (*in context of an `intArr` traversal*)
- +1** Constructs `Position` object with same row and column as identified `intArr` element
- +1** Selects constructed object when `intArr` element identified; `null` when not
- +1** Returns selected value

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

Intent: *Create a successor array based on a 2D integer array*

- +1** Creates 2D array of `Position` objects with same dimensions as `intArr`
- +1** Assigns a value to a location in 2D successor array using a valid call to `findPosition`
- +1** Determines the successor `Position` of an `intArr` element accessed by row and column (*in context of `intArr` traversal*)
- +1** Assigns all necessary locations in successor array with corresponding position object or `null` (*no bounds errors*)

Question-Specific Penalties

- 1** (s) Uses confused identifier `Arr`
- 1** (t) Uses `intArr[0].length` as the number of columns
- 1** (u) Uses non-existent accessor methods from `Position`

2017 SCORING GUIDELINES

Question 4: Scoring Notes

Part (a) findPosition			5 points
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Accesses all necessary elements of <code>intArr</code> (<i>no bounds errors</i>)		- use <code>if (...) return;</code> else <code>return null;</code> inside loop - confuse row and column bounds - fail to traverse <code>intArr</code>
+1	Identifies <code>intArr</code> element equal to <code>num</code> (<i>in context of an <code>intArr</code> traversal</i>)		use <code>.equals</code> instead of <code>==</code>
+1	Constructs <code>Position</code> object with same row and column as identified <code>intArr</code> element		- omit keyword <code>new</code> - use <code>(r,c)</code> instead of <code>Position(r,c)</code>
+1	Selects constructed object when <code>intArr</code> element identified; <code>null</code> when not	- use <code>"null"</code> instead of <code>null</code> - construct a <code>String</code> object using row and column indices	- use <code>if (...) return;</code> else <code>return null;</code> inside loop - use <code>(r,c)</code> instead of <code>Position(r,c)</code>
+1	Returns selected value		
Part (b) getSuccessorArray			4 points
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Creates 2D array of <code>Position</code> objects with same dimensions as <code>intArr</code>		omit keyword <code>new</code>
+1	Assigns a value to a location in 2D successor array using a valid call to <code>findPosition</code>	call <code>Successors.findPosition(...)</code>	- reimplement the code from <code>findPosition</code> - call <code>findPosition</code> with a single argument - call <code>this.findPosition(...)</code>
+1	Determines the successor <code>Position</code> of an <code>intArr</code> element accessed by row and column (<i>in context of <code>intArr</code> traversal</i>)	reimplement the code from <code>findPosition</code>	- call <code>findPosition</code> using an integer that is not identified with a location in <code>intArr</code> - call <code>findPosition</code> with a single argument
+1	Assigns all necessary locations in successor array with corresponding position object or <code>null</code> (<i>no bounds errors</i>)	- use <code>SuccessorArray</code> dimensions correctly, even if <code>SuccessorArray</code> was not initialized properly - only assign non-null entries to <code>SuccessorArray</code>	- reimplement the code from <code>findPosition</code> but mishandle the null case. - fail to traverse <code>intArr</code>

Return is not assessed in Part (b).

2017 SCORING GUIDELINES**Question 4: Successor Array***Part (a)*

```
public static Position findPosition(int num, int[][] intArr)
{
    for (int row=0; row < intArr.length; row++)
    {
        for (int col=0; col < intArr[0].length; col++)
        {
            if (intArr[row][col] == num)
            {
                return new Position(row, col);
            }
        }
    }
    return null;
}
```

Part (b)

```
public static Position[][] getSuccessorArray(int[][] intArr)
{
    Position[][] newArr = new Position[intArr.length][intArr[0].length];

    for (int row=0; row < intArr.length; row++)
    {
        for (int col=0; col < intArr[0].length; col++)
        {
            newArr[row][col] = findPosition(intArr[row][col]+1, intArr);
        }
    }
    return newArr;
}
```

AP[®] Computer Science A 2016 Scoring Guidelines

© 2016 The College Board. College Board, Advanced Placement Program, AP, AP Central, and the acorn logo are registered trademarks of the College Board.

Visit the College Board on the Web: www.collegeboard.org.

AP Central is the official online home for the AP Program: apcentral.collegeboard.org.

2016 GENERAL 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., writing to output, failure to compile)
- 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., 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 counter example, note that if the code declares “Bug bug;”, then uses “Bug.move()” instead of “bug.move()”, the context does **not** allow for the reader to assume the object instead of the class.*

2016 SCORING GUIDELINES

Question 1: Random String Chooser

Part (a)	RandomStringChooser	7 points
-----------------	---------------------	-----------------

Intent: *Define implementation of class to choose a random string*

- +1** Uses correct class, constructor, and method headers
- +1** Declares appropriate `private` instance variable(s)
- +1** Initializes all instance variable(s) (*point lost if parameter not used in any initialization*)
- +4** Implements `getNext`
 - +1** Generates a random number in the proper range (*point lost for improper or missing cast*)
 - +1** Chooses a string from instance variable using generated random number
 - +1** Updates state appropriately (*point lost if constructor parameter is altered*)
 - +1** Returns chosen string or `"NONE"` as appropriate

Part (b)	RandomLetterChooser	2 points
-----------------	---------------------	-----------------

Intent: *Define implementation of a constructor of a class that extends `RandomStringChooser`*

- +1** `getSingleLetters(str)`
- +1** `super(getSingleLetters(str));` (*point lost if not first statement in constructor*)

2016 CANONICAL SOLUTIONS**Question 1: Random String Chooser**

Part (a):

```
public class RandomStringChooser
{
    private List<String> words;

    public RandomStringChooser(String[] wordArray)
    {
        words = new ArrayList<String>();

        for (String singleWord : wordArray)
        {
            words.add(singleWord);
        }
    }

    public String getNext()
    {
        if (words.size() > 0)
        {
            return words.remove((int) (Math.random() * words.size()));
        }
        return "NONE";
    }
}
```

Part (b):

```
public RandomLetterChooser(String str)
{
    super(getSingleLetters(str));
}
```

2016 SCORING GUIDELINES

Question 2: Log Messages

Part (a) `LogMessage` constructor

2 points

Intent: *Initialize instance variables using passed parameter*

- +1 Locates colon
- +1 Initializes instance variables with correct parts of the parameter

Part (b) `containsWord`

2 points

Intent: *Determine whether description properly contains a keyword*

- +1 Identifies at least one properly-contained occurrence of `keyword` in `description`
- +1 Returns `true` if and only if `description` properly contains `keyword`
Returns `false` otherwise (*no bounds errors*)

Part (c) `removeMessages`

5 points

Intent: *Remove log messages containing keyword from system log list and return these messages in a new list*

- +1 Accesses all items in `messageList` (*no bounds errors; point lost if no removal attempted*)
- +1 Identifies keyword-containing entry using `containsWord`
- +1 Adds all and only identified entries to new list (*point lost if original order not maintained*)
- +1 Removes all identified entries from `messageList` (*point lost if `messageList` reordered*)
- +1 Constructs and returns new `ArrayList<LogMessage>`

2016 CANONICAL SOLUTIONS

Question 2: Log Messages

Part (a):

```
public LogMessage(String message)
{
    int colon = message.indexOf(":");
    machineId = message.substring(0, colon);
    description = message.substring(colon + 1);
}
```

Part (b):

```
public boolean containsWord(String keyword)
{
    if (description.equals(keyword))
    {
        return true;
    }
    if (description.indexOf(keyword + " ") == 0)
    {
        return true;
    }
    if (description.indexOf(" " + keyword + " ") != -1)
    {
        return true;
    }
    if (description.length() > keyword.length())
    {
        if ((description.substring(description.length() -
                                   keyword.length() - 1).equals(
                                   " " + keyword)))
        {
            return true;
        }
    }
    return false;
}
```

Part (c):

```
public List<LogMessage> removeMessages(String keyword)
{
    List<LogMessage> removals = new ArrayList<LogMessage>();

    for (int i = 0; i < messageList.size(); i++)
    {
        if (messageList.get(i).containsWord(keyword))
        {
            removals.add(messageList.remove(i));
            i--;
        }
    }
    return removals;
}
```

2016 SCORING GUIDELINES

Question 3: Crossword

Part (a)	<code>toBeLabeled</code>	3 points
-----------------	--------------------------	-----------------

Intent: Return a *boolean* value indicating whether a crossword grid square should be labeled with a positive number

- +1** Checks `blackSquares[r][c]`
- +1** Checks for black square/border above and black square/border to the left (*no bounds errors*)
- +1** Returns `true` if square should be labeled with positive number; returns `false` otherwise

Part (b)	Crossword constructor	6 points
-----------------	-----------------------	-----------------

Intent: Initialize each square in a crossword puzzle grid to have a *color* (*boolean*) and an *integer label*

- +1** `puzzle = new Square[blackSquares.length][blackSquares[0].length];`
(or equivalent)
- +1** Accesses all locations in `puzzle` (*no bounds errors*)
- +1** Calls `toBeLabeled` with appropriate parameters
- +1** Creates and assigns new `Square` to location in `puzzle`
- +1** Numbers identified squares consecutively, in row-major order, starting at 1
- +1** On exit: All squares in `puzzle` have correct color and number (*minor errors covered in previous points ok*)

Question-Specific Penalties

- 2** (p) Consistently uses incorrect name instead of `puzzle`
- 1** (q) Uses `array[].length` instead of `array[num].length`

2016 CANONICAL SOLUTIONS

Question 3: Crossword

Part (a):

```
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
{
    return (!(blackSquares[r][c]) &&
            (r == 0 || c == 0 || blackSquares[r - 1][c] ||
             blackSquares[r][c - 1]));
}
```

Part (b):

```
public Crossword(boolean[][] blackSquares)
{
    puzzle = new Square[blackSquares.length][blackSquares[0].length];
    int num = 1;

    for (int r = 0; r < blackSquares.length; r++)
    {
        for (int c = 0; c < blackSquares[0].length; c++)
        {
            if (blackSquares[r][c])
            {
                puzzle[r][c] = new Square(true, 0);
            }
            else
            {
                if (toBeLabeled(r, c, blackSquares))
                {
                    puzzle[r][c] = new Square(false, num);
                    num++;
                }
                else
                {
                    puzzle[r][c] = new Square(false, 0);
                }
            }
        }
    }
}
```

2016 SCORING GUIDELINES

Question 4: String Formatter

Part (a)	<code>totalLetters</code>	2 points
-----------------	---------------------------	-----------------

Intent: Calculate the total number of letters in a list of words

- +1** Accesses all strings in `wordList` and adds length of each to accumulated count
(no bounds errors)
- +1** Initializes and returns accumulated count

Part (b)	<code>basicGapWidth</code>	2 points
-----------------	----------------------------	-----------------

Intent: Calculate the minimum number of spaces (basic gap width) to be placed between each word in the formatted string

- +1** Calls `totalLetters` correctly and uses result
- +1** Returns correct calculated value

Part (c)	<code>format</code>	5 points
-----------------	---------------------	-----------------

Intent: Return a formatted string consisting of words from `wordList` separated by one or more spaces

- +1** Calls `basicGapWidth` and `leftoverSpaces` correctly and uses results
- +1** Adds all strings in `wordList` to formatted string in original order (no bounds errors)
- +1** Inserts *basicGapWidth* spaces between each pair of words in formatted string
- +1** Inserts one space between first *leftoverSpaces* pairs of words in formatted string
- +1** Initializes and returns formatted string (no extra or deleted characters)

2016 CANONICAL SOLUTIONS**Question 4: String Formatter**

Part (a):

```
public static int totalLetters(List<String> wordList)
{
    int total = 0;

    for (String word : wordList)
    {
        total += word.length();
    }
    return total;
}
```

Part (b):

```
public static int basicGapWidth(List<String> wordList, int formattedLen)
{
    return (formattedLen - totalLetters(wordList)) / (wordList.size()-1);
}
```

Part (c):

```
public static String format(List<String> wordList, int formattedLen)
{
    String formatted = "";
    int gapWidth = basicGapWidth(wordList, formattedLen);
    int leftovers = leftoverSpaces(wordList, formattedLen);

    for (int w = 0; w < wordList.size() - 1; w++)
    {
        formatted = formatted + wordList.get(w);
        for (int i = 0; i < gapWidth; i++)
        {
            formatted = formatted + " ";
        }
        if (leftovers > 0)
        {
            formatted = formatted + " ";
            leftovers--;
        }
    }
    formatted = formatted + wordList.get(wordList.size() - 1);

    return formatted;
}
```