2018

AP Computer Science A

Scoring Guidelines

AP® COMPUTER SCIENCE A
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

- 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.*

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

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

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

Question 1: Scoring Notes

Part (a) `simulate`			**5 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position	• use `hopDistance()` as a position, like `hopDistance() < 0`	• only use `hopDistance()` as a count, like `hopDistance() < maxHops`
+1	Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)		• do not use a loop
+1	Determines if a distance representing multiple hops is at least `goalDistance`	• use some number of hops `*` `hopDistance()` as the frog's final position	
+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	• 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 `goalDistance` after the loop • only check for starting position after the loop
Part (b) `runSimulations`			**4 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls `simulate` the specified number of times (*no bounds errors*)	• do not use the result of calling `simulate`	• do not use a loop
+1	Initializes and accumulates a count of `true` results		• initialize the count inside a loop • do not use a loop
+1	Calculates proportion of successful simulations using `double` 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 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
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.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

Question 2: Word Pair

Part (a) `WordPairList` **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) `numMatches` **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 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

Question 2: Scoring Notes

Part (a) `WordPairList`			**5 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+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`			**4 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Accesses all elements in `allPairs` (*no bounds errors*)		• access elements of `allPairs` as array elements (e.g., `allPairs[i]`)
+1	Calls `getFirst` or `getSecond` on an element from list of pairs		
+1	Compares first and second components of a pair in the list		• compare using ==
+1	Counts number of matches of pair-like values		• fail to initialize the counter

Return is not assessed in part (b).

AP® COMPUTER SCIENCE A
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 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
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

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

Question 3: Scoring Notes

Class `CodeWordChecker`			**9 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Declares header: `public class CodeWordChecker implements StringChecker`	• omit keyword `public`	• declare class `private` • declare class `static`
+1	Declares all appropriate `private` instance variables		• declare variables as `static` • omit keyword `private` • declare variables outside the class
+3	Constructors		
+1	Declares headers: `public CodeWordChecker(int __, int __, String __)` and `public CodeWordChecker(String __)`	• omit keyword `public`	• declare method `static` • declare method `private`
+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	`isValid` method		
+1	Declares header: `public boolean isValid(String __)`		• fail to declare method `public` • declare method `static`
+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 `true` if length is between min and max and does not contain the unwanted string, `false` 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 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
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.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

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()`

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

Question 4: Scoring Notes

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

Return is not assessed in Part (a).

AP® COMPUTER SCIENCE 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.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.