2018

AP CollegeBoard

AP Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free Response Question 4

- ☑ **Scoring Guideline**
- ☑ **Student Samples**
- ☑ **Scoring Commentary**

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 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()`

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.

Complete method `getColumn` below.

4Aa

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
{
    int[] output = new int[arr2D.length];
    for (int i=0; i<arr2D.length; i++)
    {
        output[i] = arr2D[i][c];
    }
    return output;
}
```

Part (b) begins on page 20

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit. 4 A6

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
{
    if (!containsDuplicates(square[0]))
    {
        for (int i=0; i< square.length; i++)
        {
            if(!hasAllValues(square[0], square[i]) ||
              !hasAllValues(square[0], getColumn(square, i))
            {
              return false;
            }
        }
        return true;
    }
    return false;
}
```


GO ON TO THE NEXT PAGE.

Complete method `getColumn` below.

4Ba

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
{
    int [] column = new int[arr2D[0].length];
    for(int i = 0; i < arr2D.length; i++)
    {
        int[i] = arr2D[i][c];
    }
    return column;
}
```

Part (b) begins on page 20

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit. 4B6

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
{
    if ( square[0].containsDuplicates() == true )
    {
        return false;
    }

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

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

    return true;
}
```


GO ON TO THE NEXT PAGE.

4Ca

Complete method `getColumn` below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
{
    int[] arrNew = new Arraylist[arr2D.length.length];

    for (i=0; i < arr2D.length; i++) {
        arrNew[i] = arr2D[i][c];
    }

    return arrNew;
}
```

Part (b) begins on page 20

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit. 4Cb

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

```
int[] arr1 = square[0];

if (!containsDuplicates) {
    for (i=0; i < square.length; i++) {
        int[] arr2 = square[i];
        if (hasAllValues) {
            for (j=0; j < square.length.length; j++) {
                int[] arr3 = new ArrayList(square.length];
                for (z=0; z < square.length; z++) {
                    arr3.add(square[j][z]);
                }
                arr3 = arr2;
                if (hasAllValues) {
                    return True;
                }
            }
        }
    }
}
return false;
```

AP® COMPUTER SCIENCE A
2018 SCORING COMMENTARY

Question 4

Overview

This question tested the student's ability to:

- Write program code to create, traverse, and manipulate elements in 1D array or `ArrayList` objects;
- Write program code to create, traverse, and manipulate elements in 2D array objects; and
- Write program code to create objects of a class and call methods.

The students were expected to write two static methods of an enclosing `ArrayTester` class. Additionally, students were required to use method `getColumn` from part (a) and two already-implemented methods of the `ArrayTester` class, `containsDuplicates` and `hasAllValues`, in their part (b) solutions.

In part (a) students were asked to construct a one-dimensional array and copy the items from a specified column of a given two-dimensional array into the new array. Students were expected to be able to construct an array with the correct number of elements, which is the number of rows in the two-dimensional array, not the number of columns. Once the array was constructed, students were expected to write a loop that accesses each item in the given column and assigns it to the corresponding element of the new array.

In part (b) students were given a square two-dimensional array and asked to evaluate if the two-dimensional array was a Latin square. A two-dimensional array of integers is a Latin square if:

- The first row has no duplicates;
- All values in the first row of the square appear in every row of the square; and
- All values in the first row of the square appear in every column of the square.

Sample: 4A
Score: 9

In part (a) an `int` array of size `arr2D.length` is correctly constructed. The response earned point 1. A loop that executes the correct number of times is used to access all items in one column of `arr2D`, which earned point 2. All items in a column of `arr2D` are assigned to the corresponding elements of the constructed array, which earned point 3. At the end of the method, the constructed array contains all items from the specified column in `arr2D` in the correct order. The response earned point 4. Part (a) earned 4 points.

In part (b) the first row of `square` is checked for duplicates with a correct call to method `containsDuplicates`. The response earned point 5. A correct `for` loop is used to access all rows and columns of `square`. The `hasAllValues` method determines if every row and column of `square` contains all the elements of the first row of `square`. The response earned point 6 and point 7. A correct call to method `getColumn` is used, which earned point 8. All three Latin square conditions are satisfied. The response earned point 9. Part (b) earned 5 points.

Sample: 4B
Score: 6

In part (a) there is an attempt to construct an `int` array of size `arr2D.length;` however, the response incorrectly uses `arr2D[0].length`. The response did not earn point 1. A correct `for` loop is used to access all items in one column of `arr2D`, which earned point 2. Items in a column of `arr2D` are not assigned to the corresponding elements of the constructed array. The response incorrectly assigns the items to `int[i]` instead

Question 4 (continued)

of `column[i]`. The response did not earn point 3. Going forward, it is assumed that all accessed items are correctly assigned to the constructed array. At the end of the method, the constructed array contains all items from the specified column in `arr2D` in the correct order. The response earned point 4. Part (a) earned 2 points.

In part (b) the first row of `square` is checked for duplicates with a syntactically incorrect call to the method `containsDuplicates`. The response earned point 5; however, a question-specific penalty of 1 point was assessed for the syntactically incorrect call. Correct `for` loops are used to access all rows and columns of `square`. The `hasAllValues` method determines if every row and column of `square` contains all the elements of the first row of `square`. The response earned point 6 and point 7. A correct call to method `getColumn` is used, which earned point 8. All three Latin square conditions are satisfied. The response earned point 9. Part (b) earned 4 points.

Sample: 4C
Score: 3

In part (a) the response does not correctly construct an `int` array of size `arr2D.length`. The response constructs the array as an `ArrayList[arr2D.length.length]`. The response did not earn point 1. Going forward, it is assumed that the array is properly constructed. A loop that executes the correct number of times is used to access all items in one column of `arr2D`, which earned point 2. All items in a column of `arr2D` are assigned to the corresponding elements of the constructed array, which earned point 3. At the end of the method, the constructed array contains all items from the specified column in `arr2D` in the correct order. The response earned point 4. Part (a) earned 3 points.

In part (b) there is an invalid call to method `containsDuplicates` that does not include a parameter. The response did not earn point 5. There is an invalid call to method `hasAllValues` that does not include any parameters, so the question-specific syntax penalty cannot be applied. The response did not earn point 6 and point 7. There is no call to method `getColumn`. The response did not earn point 8. Some of the Latin square conditions are incorrectly tested. The response did not earn point 9. Part (b) earned 0 points.