2022

AP CollegeBoard

AP® Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free-Response Question 4

- ☑ **Scoring Guidelines**
- ☑ **Student Samples**
- ☑ **Scoring Commentary**

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 (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`; with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "ArayList" instead of "ArrayList". As a counterexample, note that if the code declares `"int G=99, g=0;"`, then uses `"while (G < 10)"` instead of `"while (g < 10)"`, the context does **not** allow for the reader to assume the use of the lower case variable.*

Question 4: 2D Array — 9 points

Canonical solution

(a) **4 points**

```
public void repopulate()
{
   for (int row = 0; row < grid.length; row++)
   {
      for (int col = 0; col < grid[0].length; col++)
      {
         int rval = (int)(Math.random() * MAX) + 1;
         while (rval % 10 != 0 || rval % 100 == 0)
         {
            rval = (int)(Math.random() * MAX) + 1;
         }
         grid[row][col] = rval;
      }
   }
}
```

(b) **5 points**

```
public int countIncreasingCols()
{
   int count = 0;

   for (int col = 0; col < grid[0].length; col++)
   {
      boolean ordered = true;

      for (int row = 1; row < grid.length; row++)
      {
         if (grid[row][col] < grid[row-1][col])
         {
            ordered = false;
         }
      }

      if (ordered)
      {
         count++;
      }
   }

   return count;
}
```

(a) `repopulate`

	Scoring Criteria	Decision Rules	
1	Traverses `grid` (*no bounds errors*)	Responses **will not** earn the point if they • fail to access an element of `grid` • access the elements of `grid` incorrectly • use enhanced `for` loops without using a `grid` element inside the loop	**1 point**
2	Generates a random integer in a range based on `MAX`	Responses **can** still earn the point even if they • assume or verify that `MAX >= 10` Responses **will not** earn the point if they • fail to cast to an `int`	**1 point**
3	Ensures that all produced values are divisible by 10 but not by 100	Responses **can** still earn the point even if they • fail to use a loop	**1 point**
4	Assigns appropriate values to all elements of `grid` (*algorithm*)	Responses **can** still earn the point even if they • assume or verify that `MAX >= 10` • produce some values that are not divisible by 10 or divisible by 100, if the range and distribution are otherwise correct Responses **will not** earn the point if they • use enhanced `for` loops and fail to maintain indices • produce values that are not equally distributed • produce values outside the specified range • exclude values that should be considered valid (other than errors in 10/100 handling)	**1 point**
		Total for part (a)	**4 points**

(b) `countIncreasingCols`

	Scoring Criteria	Decision Rules	
5	Traverses `grid` in column major order (*no loop header bounds errors*)	Responses **can** still earn the point even if they • access an out-of-bounds row or column index adjacent to the edge of the grid, if the loop bounds include only valid indices Responses **will not** earn the point if they • fail to access an element of `grid` • access the elements of `grid` incorrectly	**1 point**
6	Compares two elements in the same column of `grid`	Responses **can** still earn the point even if they • access elements of `grid` incorrectly • access elements in nonadjacent rows • compare elements with `==` • compare two elements in the same row instead of the same column	**1 point**
7	Determines whether a single column is in increasing order (*algorithm*)	Responses **can** still earn the point even if they • fail to reset variables in the outer loop before proceeding to the next column Responses **will not** earn the point if they • fail to access all pairs of adjacent elements in a single column • cause a bounds error by attempting to compare the first element of a column with a previous element or the last element of a column with a subsequent element • incorrectly identify a column with at least one pair of adjacent elements in decreasing order as increasing	**1 point**
8	Counts all columns that are identified as increasing (*algorithm*)	Responses **can** still earn the point even if they • detect increasing order for each row instead of each column • incorrectly identify increasing columns in the inner loop Responses **will not** earn the point if they • fail to initialize the counter	**1 point**

		• fail to reset variables in the outer loop causing subsequent runs of the inner loop to misidentify columns	
9	Returns calculated count of increasing columns	Responses **can** still earn the point even if they • calculate the count incorrectly	**1 point**
		Total for part (b)	**5 points**
	Question-specific penalties		
	None		
		Total for question 4	**9 points**

Alternate Canonical for Part (a)

```
public void repopulate()
{
   for (int row = 0; row < grid.length; row++)
   {
      for (int col = 0; col < grid[0].length; col++)
      {
         int rval = ((int)(Math.random() * (MAX / 10)) + 1) * 10;
         while (rval % 100 == 0)
         {
            rval = ((int)(Math.random() * (MAX / 10)) + 1) * 10;
         }
         grid[row][col] = rval;
      }
   }
}
```

Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1 ○ Question 2 ○ Question 3 ○ Question 4 ●

Begin your response to each question at the top of a new page.

```
// Part A

Public void repopulate() {
for(int r = 0; r < grid.length; r++) {
 for(int c=0; c<grid[0].length; c++){
  int rNum = 0
  while(rNum%10 != 0 || rNum%100 == 0) {
    rNum= (int)(Math.random * MAX) + 1;
   }
   grid[r][c] = rNum;
}
}
}

// Part B

Public int countIncreasingCols() {
int count = 0;
for(int c = 0; c < grid[0].length; c++) {
 for(int r = 0; r < grid.length - 1; r++) {
   int ColCount = 0;
   if(grid[r+1][c] > grid[r][c]) {
     ColCount++)
   }
}  if(ColCount == grid.[0].length) {
    count ++;
```

Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1	Question 2	Question 3	Question 4
○	○	○	●

Begin your response to each question at the top of a new page.

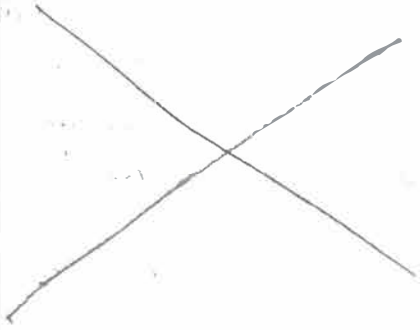
```
}
  return count;
}
```


Use a pencil only. Do NOT write your name. Do NOT write outside the box.

Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1	Question 2	Question 3	Question 4
○	○	○	●

Begin your response to each question at the top of a new page.



```
//part a
public void repopulate(){
  int val=0;
  int check1=0;
  int check2=0;
  for(int x=0; x<grid.length(); x++){
    for(int i=0; i<grid[x].length(); i++){
      do{
        val = Math.random(0, MAX+1);
        check1= val%10
        check2= val%100
      } while((check1 == 0)&&(check2 !=0));
      grid[x][i] = val;
    }
  }
}
```


Use a pencil only. Do NOT write your name. Do NOT write outside the box.

Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1 ○ Question 2 ○ Question 3 ○ Question 4 ●

Begin your response to each question at the top of a new page.

```
// part b

public int countIncreasingCols(){
   int count=0;
   int num=0;
   for (int c=0; c<grid[0].length(); c++){
      for(int r=1; r< grid.length(); r++){
         if(grid[r][c] > grid[r-1][c]){
            count ++;
         }
      }
      if(count== grid.length()){
         num++;
      }

   return num;
}
```

Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1 ○ Question 2 ○ Question 3 ○ Question 4 ●

Begin your response to each question at the top of a new page.

a)

```
{
  private int num;
  for(int r = 0; r < grid.length(); r++)
     for (int c = 0; c < grid[0].length();c++)
     {
       num = (int)(Math.random() * Max + 1)
       if(!(num %10 == 0)) num += 10;
        grid[r][c] = num;
     }
}
```

b)

```
{
 private int count;
 for(int r = 0; r < grid.length(); r++)
 {
    if (grid[0].length() == 1) count++;
    else for(int c = 1; c < grid[0].length; c++)
    {
      if (grid[c-1] < grid[c]) count++;
    }
 }
 return count;
}
```

Question 4

Note: Student samples are quoted verbatim and may contain spelling and grammatical errors.

Overview

This question tested the student's ability to:

- Write program code to call methods.
- Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements.
- Write program code to traverse and manipulate elements in 2D array objects.

This question involved the manipulation of a two-dimensional (2D) array of `int` values. A class that included two methods, one written in part (a) and one written in part (b), was provided.

In part (a) students were asked to write a `void` method, `repopulate`, that assigned newly generated random numbers to each element of the 2D array instance variable, `grid`. The new elements in the array must be between 1 and the class constant `MAX`, inclusive; must be divisible by 10; and must not be divisible by 100. All values must have an equal chance of being generated. Students were expected to traverse all elements of the array and assign a newly generated random number satisfying each of the criteria to each array element.

In part (b) students were asked to write a method, `countIncreasingCols`, that returned the number of columns in the 2D array instance variable, `grid`, that are in increasing order. A column is in increasing order when each element of the column after the first row is greater than or equal to the element of the column in the previous row. Students were expected to traverse the array in column-major order. Students were expected to identify columns in which each pair of adjacent elements satisfy the increasing criterion. Students were then expected to count the identified columns and return the count.

Sample: 4A
Score: 8

In part (a) point 1 was earned because the loop variables `r` and `c` will reach every pair of row and column indices in `grid`, and values are accessed using `grid[r][c]` in the body of the inner loop. To earn this point, a response must use nested loops to visit every position in the array `grid`, with no bounds errors and no omissions. Point 2 was earned because a correct computation is used to obtain a random `int` value in the range [`1`, `MAX`]. The missing `()` on the parameter-less method invocation `random` is one of the minor errors for which no penalty is assessed on this exam. (See the "No Penalty" category on page 1 of the Scoring Guidelines for a complete list.) This point focuses on generating a floating-point random number, scaling the range of possible returned numbers, and converting the number to an `int`. A response can still earn the point if the computation produces an incorrect range of integers as long as the computation includes a reference to the constant `MAX`, a cast to `int`, and a correct call to `Math.random`. Point 3 was earned because a correct condition is used in the `while` loop to identify numbers that are not divisible by 10 or are divisible by 100. This point is earned when the response checks these conditions correctly, even if no loop is present. A response

Question 4 (continued)

earning this point may include various combinations of `==`, `!=`, `&&`, and `||`. Conditions such as `(val % 10 == 0 && val % 100 != 0)`, its logical equivalent, or its logical reverse could all be correct, depending on the context. A common approach is to use a `while` loop to repeatedly generate new random numbers until the necessary conditions are met. Point 4 was earned because the entire array `grid` is filled with correctly generated values. To earn this point, a response must ensure that values are within the range [`1`, `MAX`], are stored in every position of the array `grid`, and occur with equal probability. Responses that do not properly cast each random number to an `int` or do not correctly ensure that all produced values are divisible by 10 but not by 100 may still earn the point. However, a random number generated without using `MAX` will not earn this point.

In part (b) point 5 was earned because the traversal is in column-major order, and all elements of the array `grid` are accessed without any bounds errors. This point focuses on column-major traversal, so a reference to `grid[x][y]` requires an outer loop that generates `y` indices and an inner loop that generates `x` indices. Responses that interchange the loops (row-major order) or do not access all elements of `grid` will not earn this point. Responses that go out of bounds at the edge of `grid` can still earn this point if the loop bounds produce only valid indices of `grid`. Point 6 was earned. Any solution that compares two elements of the same row or column by fixing one index and varying the other by addition or subtraction, even if the addition or subtraction results in an out-of-bounds index, will earn point 6. Also, any comparison operator (`>`, `>=`, `<`, `<=`, `==`, `!=`), even if it is not the correct comparison for the algorithm, can be used to earn this point. Point 7 was not earned for multiple reasons: 1) the comparison `grid[r + 1][c] > grid[r][c]` should use `>=` in order to be logically correct; 2) the column-counting variable `colCount` is local to the inner loop so does not retain its value across columns, nor is it in scope in the outer loop, where its value is checked against `grid[0].length`; 3) the comparison with `colCount` in the outer loop should be against `grid[0].length - 1` since an array with N rows has only N - 1 unique pairs of elements per column; and 4) if the array contains only 1 row, the inner loop will not iterate, resulting in no columns correctly identified and counted. A response could still earn this point should it fail to reset necessary variables before proceeding to the next column; however, it must successfully maintain the count for at least one entire column. A response will also not earn this point if a bounds error is caused by attempting to compare to an element with an index beyond the bounds of the array. Point 8 was earned because the algorithm for counting increasing columns initializes a counter (`count`), increments it when an increasing column is identified, and resets any variables used to identify increasing columns (`colCount`) at the beginning of each column. Incorrect placement of the reset (in this case, in the inner loop instead of the outer loop) was assessed in point 7 and does not prevent point 8 from being earned. Point 8 can also still be earned if the algorithm in a response counts increasing rows instead of increasing columns, even if the algorithm incorrectly identifies increasing columns (or rows). Point 9 was earned because the variable `count`, which is used to calculate the number of increasing columns, is returned. To earn this point, the response must return an integer that is the result of a calculation, but the calculation need not be correct. For responses that use more than one integer variable, the response must return the variable that contains the column count to earn this point.

Question 4 (continued)

Sample: 4B
Score: 4

In part (a) point 1 was earned because a nested loop iterates through the entire 2D array, visiting each element without any bounds errors. Rows are represented by `x` and columns by `i` in the response. The array is accessed inside the inner loop correctly using `x` and `i`, so all valid locations are accessed. Point 2 was not earned because parameters are erroneously introduced for the `random` method. Also, there is no cast to `int`, nor a calculation using `MAX`. Any of these reasons is sufficient to not earn this point. Point 3 was not earned because the logic is reversed—a correct condition would be `check1 != 0 || check2 == 0`. This response incorrectly generates random numbers that are not multiples of 10, as well as numbers that are multiples of 100. Point 4 was not earned because `val` is not the result of a computation using `MAX`, nor does the response ensure that values are equally distributed or in the range [`1`, `MAX`].

In part (b) point 5 was earned because `grid` is traversed in column-major order with no bounds errors. Rows are represented by `r` and columns by `c`. The loop for the rows starts at `1`, so there is not a bounds error when `grid[r - 1][c]` is accessed. Point 6 was earned because two adjacent column elements are compared, `grid[r][c]` and `grid[r - 1][c]`. Point 7 was not earned because the comparison uses `>` rather than `>=`. Additionally, the response checks if `count == grid.length`. Either of these errors would cause the response to not earn this point. Point 8 was not earned because the variable `count` is not reset before each execution of the inner loop. Therefore, no more than one increasing column can be identified. The missing `}` to close the outer `for` loop is a no-penalty error, as indentation clearly conveys intent. Point 9 was earned because a calculated count is returned, represented by the variable `num`. Had `count` been returned, this point would not have been earned as the calculated value returned must represent the number of increasing columns.

Sample: 4C
Score: 3

In part (a) point 1 was earned because a nested loop iterates through the entire 2D array, visiting each element without any bounds errors. Rows are represented by `r` and columns by `c` in the response. The array is correctly accessed inside the inner loop using `r` and `c`, so all valid locations are accessed. Point 2 was earned because a random integer is generated based on `MAX` with a correct call to `Math.random` and a correct cast to an `int` value. Point 3 was not earned because the response does not check that the generated random integer is not divisible by 100. The `private` qualifier on the local variable `num` is one of the minor errors for which no penalty is assessed on this exam. Point 4 was not earned because the generated random numbers are outside the specified range [`1`, `MAX`]. The expression `num *= 10` inside the `if` statement causes some numbers in the range [`MAX + 1`, `10 * MAX`] to be generated as well.

In part (b) point 5 was not earned because the column index is not used in accessing elements, and also because the outer loop doesn't generate column indices, so the traversal is not in column-major order. Thus, there is no access of an `int` element of the `grid` 2D array. A reference to `grid` with two indices is required. Point 6 was not earned because there is no comparison of two adjacent elements of `grid`. The response compares `grid[c - 1]` to `grid[c]`, as though these were

Question 4 (continued)

adjacent `int` values, when they are, in fact, 1D arrays and cannot be compared in this way. Point 7 was not earned because a single column is not identified as increasing. Point 8 was not earned because `count` is based on the number of comparisons of adjacent elements instead of the number of increasing columns. Point 9 was earned because a calculated count is returned. The correctness of this value does not affect earning this point.