2025

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.*

2025 Digital Decision Rules

- Some non-ASCII characters are not printing correctly in ONE, particularly extended punctuation. Certain kinds of double-quotes may display as â□□; a character that displays as ï¼□ may be a parenthesis, comma, or semicolon (or other punctuation). If the badly displayed character makes sense as one of those, evaluate the response accordingly.
- If there are missing closed-double-quotes or closed-parentheses, assume they are at the end of the line where they opened, immediately before the semicolon or curly bracket (if any).
- Assume an open/left curly bracket `{` immediately after any method header or class header that does not already have one.
- Assume an appropriate amount of closing brackets before any method header to close all open brackets from the previous method or constructor.
- If there are missing curly brackets, clear indentation can "convey intent". Evaluate the response accordingly.
- Inside a method with left-justified code, indentation cannot "convey intent", so missing curly brackets cannot be assumed. Without bracketing or indentation, only the first line of a `while` / `if` / `for` is controlled by the statement; with open curly bracket and no indentation cues, the entire remainder of the method is "inside" the statement.

No Penalty

- `:` instead of `;` and vice versa
- `,` instead of `;` and vice versa

Question 4: 2D Arrays — 9 points

Canonical solution

a. **4 points**

```
public SumOrSameGame(int numRows, int numCols)
{
   puzzle = new int[numRows][numCols];

   for (int row = 0; row < numRows; row++)
   {
      for (int col = 0; col < numCols; col++)
      {
         puzzle[row][col] = (int)(Math.random() * 9) + 1;
      }
   }
}
```

b. **5 points**

```
public boolean clearPair(int row, int col)
{
   int val1 = puzzle[row][col];
   for (int currRow = row; currRow < puzzle.length; currRow++)
   {
      for (int currCol = 0; currCol < puzzle[0].length; currCol++)
      {
         int val2 = puzzle[currRow][currCol];
         if (currRow != row || currCol != col)
         {
            if (val1 == val2 || val1 + val2 == 10)
            {
               puzzle[row][col] = 0;
               puzzle[currRow][currCol] = 0;
               return true;
            }
         }
      }
   }
   return false;
}
```

a. `SumOrSameGame`

	Scoring Criteria	Decision Rules	
1	Constructs correctly sized 2D array of `int` and assigns to instance variable `puzzle`	Responses **will not** earn the point if they • fail to update the instance variable `puzzle`, e.g., by redeclaring as a local variable • print or return a value instead of or in addition to updating `puzzle`	**1 point**
2	Traverses 2D array (*no bounds errors*)	Responses **can** still earn the point even if they • fail to construct the 2D array, as long as traversal uses correct bounds • fail to assign to the 2D array elements Responses **will not** earn the point if they • fail to access an element of the 2D array in the traversal	**1 point**
3	Generates a random integer that is uniform in the range `[1, 9]`	Responses **will not** earn the point if they • fail to call `Math.random` or an equivalent method • make any incorrect call to `Math.random` or an equivalent method • call `random` without `Math.` • fail to cast to an `int`	**1 point**
4	Assigns values to 2D array elements (*algorithm*)	Responses **can** still earn the point even if they • generate the random value incorrectly • assign to a local 2D array instead of the instance variable • fail to assign some elements due to a bounds error Responses **will not** earn the point if they • assign a value other than the attempted randomly generated value • correctly initialized the 2D array but now assign somewhere else • assign the same value to all visited elements • fail to assign to all visited elements	**1 point**

b. `clearPair`

	Scoring Criteria	Decision Rules	
5	Accesses all and only necessary elements of `puzzle` in rows `>=` `row` (*no bounds errors*)	Responses **can** still earn the point even if they • access additional rows, as long as they also guard against pairing them • pair the element at `row` and `col` with itself • omit extra elements due to incorrect self-pairing guard, as long as the bounds would otherwise support accessing all necessary elements • return early, as long as bounds and indices would otherwise support accessing all necessary elements Responses **will not** earn the point if they • fail to access elements of `puzzle` correctly	**1 point**
6	Guards against pairing an element with itself	Responses **will** not earn the point if they • omit extra potential pairs with their self-pairing guard • have no loop	**1 point**
7	Determines whether two elements are the same or sum to 10	Responses **can** still earn the point even if they • pair the element at `row` and `col` with itself • have no loop Responses **will not** earn the point if they • test only one of the conditions	**1 point**
8	Sets a 2D array element to `0` *(in the context of a loop)*	Responses **can** still earn the point even if they • set only one identified element to `0` • incorrectly identify elements • set more than 2 elements to `0`	**1 point**
9	Sets identified elements to `0` when match is found and returns appropriate `boolean` in both cases (*algorithm*)	Responses **can** still earn the point even if they • identify elements incorrectly, as long as there is some conditional selection Responses **will not** earn the point if they • set only one identified element to `0` • change more than 1 element besides the element at `row` and `col` • return an incorrect value due to an early return • return an incorrect value due to not returning after clearing once	**1 point**
	Question-specific penalties		
	None		

Part (a)

```
public SumOrSameGame( int numRows, int numCols ){

   int[][] matrix = new int[numRows][numCols];
   puzzle = matrix;

   for( int i = 0; i < numRows; i++){
      for( int j = 0; j < numCols; j++){
         puzzle[i][j] = (Math.random()*9)+1;
      }
   }

}
```

Part (b)

```
public boolean clearPair( int row, int col){

   int val = puzzle[row][col];

   for(int i = row; i < puzzle.length; i++){
      for( int j = 0; j < puzzle[0].length; j++){
         if( i >= row ){
               if( (puzzle[i][j] == val) && !(i == row && j == col){
                  puzzle[i][j] = 0;
                  puzzle[row][col] = 0;
                  return true;
               }
               if( (puzzle[i][j] + val == 10) && !(i == row && j == col){
                  puzzle[i][j] = 0;
                  puzzle[row][col] = 0;
                  return true;
               }
         }
      }
   }

   return false;
}
```

Part (a)

```
public SumOrSameGame(int numRows, int numCols){
   int[][] puzzle = new int[numRows][numCols];
   for(int r = 0; r < numRows; r++){
      for(int c = 0; c < numCols; c++){
         int i = (int)(Math.random()*8)+1;
         puzzle[r][c] = i;
      }
   }
}
```

Part (b)

```
public boolean clearPair(int row, int col){
   boolean pair = false;
   for(int i = row; i < puzzle.length; i++){
      for(int c = col; c < puzzle[0].length; c++){
         if(puzzle[i][c] == puzzle[row][col]){
            puzzle[i][c] == 0;
            puzzle[row][col] == 0;
            pair = true;
         }
         if(puzzle[i][c]+puzzle[row][col] == 10){
            puzzle[i][c] == 0;
            puzzle[row][col] == 0;
            pair = true;
         }
      }

   }
   return pair;
}
```

Part (a)

```
public SumOrSameGame(int numRows, int numCols){
   int[][] puzzle = new ArrayList<>;
   for(int i = 0; i =< numRows; i++){
      for(int k = 0; k =< numCols; k++){
         puzzle[i][k] = (int)(Math.random() * 10) + 1;
      }
   }

}
```

Part (b)

```
public boolean clearPair(int row, int col){
    for(int j = 0; j > puzzle.length(); j++){
       for(int p = 0; p > puzzle[0].length(); p++){
          if(puzzle[row][col] != puzzle[j][p]{
            return false;
          }
          else{
             puzzle[j][p] = 0;
             return true;
          }
       }
    }


}
```

Question 4

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

Overview

NEW for 2025: The question overviews can be found in the *Chief Reader Report on Student Responses* on AP Central.

Sample: 4A
Score: 8

In part (a) point 1 was earned by constructing the 2D array using `new int[numRows][numCols]` and assigning it to the instance variable `puzzle`. To earn this point, the construction must use the `new` keyword with the proper element type for the 2D array and have both dimensions match the corresponding parameters to the constructor. The newly constructed array is initially assigned to `matrix`, which is a local variable, then `puzzle` is assigned to also reference the new array. Assignment only to a local variable would not have been sufficient for this point. Point 2 was earned because the nested `for` loops traverse all elements of the 2D array without a bounds error, accessing `puzzle[i][j]` within the loop. Point 3 was not earned because although the response calls `Math.random()` correctly, multiplies by `9` to create a number in the range [0,9), and does the required addition to shift the range to start at `1`, it does not cast the result to an `int` as required to assign into the 2D array element. Point 4 was earned with the assignment statement `puzzle[i][j] = (Math.random()*9)+1;` within the loop. To earn this point, each element of the 2D array must be assigned its own randomly generated value. This point was earned even though the generated number is not correct due to the missing cast, as that was assessed in Point 3.

In part (b) point 5 was earned because the response iterates through all the necessary elements of `puzzle`. To earn this point, the traversal must visit all columns in rows starting at index `row` without a bounds error and access the elements of the 2D array correctly. The check that `i` is greater than or equal to `row` is unnecessary because the loop starts at `row` but would have been needed if `row` were initialized to `0`. Point 6 was earned because the response guards against pairing `puzzle[row][col]` with itself in all cases. The Boolean expression `!(i == row && j == col)` is logically equivalent to the canonical solution's `(currRow != row || currCol != col)` using De Morgan's law. Point 7 was earned because the response determines whether two elements are the same, using `==` in the first nested `if`, or sum to 10, in the second nested `if`. The separate conditions are logically equivalent to the `||` in the innermost `if` in the canonical solution. Point 8 was earned by setting `puzzle[i][j] = 0`. This response correctly sets both elements but setting either element from an identified pair is sufficient for this point. Point 9 was earned because the response clears both elements of an identified pair and returns `true`, or returns `false` if no pair was identified. Each of the two inner `if` statements assigns both `puzzle[i][j] = 0` and `puzzle[row][col] = 0` and then returns `true`. The response returns `false` outside the loops, after all possible pairs have been considered and none has been found.

Question 4 (continued)

Sample: 4B
Score: 5

In part (a) point 1 was not earned because the reference to the correctly constructed new 2D array is assigned to `int[][] puzzle`, which is a local variable, not the instance variable `puzzle`. Point 2 was earned with the nested `for` loops and access to `puzzle[r][c]` within the loops, which will access all the elements of the 2D array without a bounds error. This point was earned even though `puzzle` is the local variable, not the instance variable, which was assessed in Point 1. Point 3 was not earned because `(int)(Math.random()*8)+1` creates a random integer in the range from 1 through 8, inclusive, not 1 through 9, inclusive. Point 4 was earned with the assignment `puzzle[r][c] = i;` within the nested loops. This point was earned even though the generated number is not correct, as that was assessed in Point 3.

In part (b) point 5 was not earned because the inner loop always begins with the index determined by the parameter `col` instead of index zero. Thus, no elements in earlier columns will be considered. Point 6 was not earned because the response does not contain any code that prevents comparing `puzzle[row][col]` with itself. Point 7 was earned because the response checks both conditions in the two separate `if` statements. Point 8 was earned by setting `puzzle[i][c] = 0` and `puzzle[row][col] = 0`. Setting either element from an identified pair is sufficient for Point 8. Point 9 was earned because the response clears an identified pair and returns `true`, or returns `false` if no pair was identified. In this response, each of the two inner `if` statements assigns both `puzzle[i][c]` and `puzzle[row][col]` to `0` and sets `pair = true`. After the loops end, the response will return `true` if a pair was identified. If no pair was identified and thus `pair` remains unchanged, `false` is returned. Although the loops continue to run after a pair is cleared, no additional changes to the values in the array will be made because the code that finds a match uses the updated value at `puzzle[row][col]` to perform the comparisons, which is now `0`. Thus, there can be no match based on summing to `10` (since all the values in the array are now between `0` and `9`) and any match based on having the same value will be a match with another element whose value is also already `0`, so re-setting both to `0` will not result in a change.

Sample: 4C
Score: 2

In part (a) point 1 was not earned for the following reasons, either of which would cause the response not to earn Point 1. First, `int[][] puzzle` is a local variable, not the instance variable `puzzle`, and second, the constructed object is of type `ArrayList`. Point 2 was not earned because each loop will execute one too many times due to the relational operator `<=` (incorrectly entered as `=<`) in the loop sustaining condition, and thus `puzzle[i][k]` will access the 2D array out of bounds on the last iteration. Note that `puzzle[i][k]` could be considered valid access, in general, even though the construction was incorrect. The construction is assessed in Point 1. Point 3 was not earned, because the random number generation `(int)(Math.random() * 10) + 1` will produce an `int` in the range `1` through `10`, inclusive, instead of `1` through `9`, inclusive. Point 4 was earned with the assignment of the random number into `puzzle[i][k]` within the loop.

Question 4 (continued)

In part (b) point 5 was not earned for the following reasons, either of which would cause the response not to earn Point 5. The first reason is that there are zero iterations of the loops because the loop sustaining conditions `j > puzzle.length()` and `p > puzzle[0].length()` use the `>` relational operator instead of `<`. The second reason is that the outer `for` loop control variable `j` always begins at row zero and does not guard against processing rows with indices less than the parameter `row`. For arrays `length` is a field, not a method, and thus should be `puzzle.length` not `puzzle.length()`. This is an example of one of the minor errors for which no penalty is assessed, as noted on the *Applying the Scoring Criteria* page found in the Scoring Guidelines. Point 6 was not earned because the response does not contain any code that prevents comparing `puzzle[row][col]` with itself. Point 7 was not earned because the response does not test for both conditions. There is no test for whether the elements sum to `10`. The missing closing parenthesis after `puzzle[j][p]` in the conditional is an example of a minor no penalty error. Missing closing parentheses should be assumed before a subsequent `{` (as in this response) or at the end of the line. Point 8 was earned with the statement `puzzle[j][p] = 0;` in the `else` clause within the loop. Clearing a single element of the identified pair is sufficient for Point 8. Point 9 was not earned for the following reasons, either of which would cause the response not to earn Point 9. First, the response returns `false` from the loop prematurely when a match is not immediately found. Secondly, only one value is cleared when a match is found. Although the value at `puzzle[j][p]` is set to `0` when a match is found, the value at `puzzle[row][col]` is not. Both values must be cleared to earn Point 9.