2023

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 Arrays — 9 points

Canonical solution

(a) — **4 points**

```
public boolean moveCandyToFirstRow(int col)
{
   if (box[0][col] != null)
   {
      return true;
   }

   for (int row = 1; row < box.length; row++)
   {
      if (box[row][col] != null)
      {
         box[0][col] = box[row][col];
         box[row][col] = null;
         return true;
      }
   }

   return false;
}
```

(b) — **5 points**

```
public Candy removeNextByFlavor(String flavor)
{
   for (int row = box.length - 1; row >= 0; row--)
   {
      for (int col = 0; col < box[0].length; col++)
      {
         if (box[row][col] != null &&
             box[row][col].getFlavor().equals(flavor))
         {
            Candy taken = box[row][col];
            box[row][col] = null;
            return taken;
         }
      }
   }
   return null;
}
```

(a) `moveCandyToFirstRow`

	Scoring Criteria	Decision Rules	
1	Accesses all necessary elements of column `col` of `box` (*no bounds errors*)	Responses **can** still earn the point even if they • return early, as long as the loop bounds are appropriate Responses **will not** earn the point if they • fail to access an element of `box` in the loop • access the elements of `box` incorrectly	**1 point**
2	Compares candy box element at row `0` and column `col` to `null`	Responses **can** still earn the point even if they • make the comparison inside the loop or at some incorrect point in the code Responses **will not** earn the point if they • fail to use `!=` or equivalent	**1 point**
3	Identifies and moves appropriate `Candy` object to first row if necessary (*algorithm*)	Responses **can** still earn the point even if they • return early, as long as the identify-and-move are inside a loop and would work if the loop got that far Responses **will not** earn the point if they • fail to replace the moved `Candy` object with `null` • move or swap objects when the first row is already occupied	**1 point**
4	Returns `true` when non-empty square is identified and `false` if non-empty square is not identified in the context of a loop (*algorithm*)	Responses **can** still earn the point even if they • fail to replace the moved `Candy` object with `null` • incorrectly identify a non-empty square Responses **will not** earn the point if they • return early	**1 point**
		Total for part (a)	**4 points**

(b) `removeNextByFlavor`

	Scoring Criteria	Decision Rules	
5	Traverses `box` in specified order (bottom to top, left to right) (*no bounds errors*)	Responses **will not** earn the point if they • fail to access an element of `box` in the loop • access the elements of `box` incorrectly	**1 point**
6	Guards against a method call on a `null` element of the candy box (*in the context of an* `if` *statement*)	Responses **will not** earn the point if they • fail to use `!=` or equivalent	**1 point**
7	Calls `getFlavor` on a `Candy` object	Responses **can** still earn the point even if they • access the element of the candy box incorrectly • call `getFlavor` on the incorrect `Candy` object Responses **will not** earn the point if they • call `getFlavor` on an object of a different type or on the `Candy` class • attempt to create a `Candy` object • include parameters	**1 point**
8	Compares a `Candy` object's flavor with `flavor` parameter	Responses **will not** earn the point if they • fail to use `equals`	**1 point**
9	Replaces first matching `Candy` object with `null` and returns replaced object (*algorithm*)	Responses **can** still earn the point even if they • access elements of the candy box incorrectly • call `getFlavor` incorrectly or on a `null Candy` object • compare a `Candy` object's flavor to the parameter using `==` Responses **will not** earn the point if they • fail to replace the identified object within the 2D array with `null` before returning • fail to return `null` when no matching `Candy` object is found • set multiple elements to `null`	**1 point**
		Total for part (b)	**5 points**

Question-specific penalties
None

Total for question 4 **9 points**

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.

```
public boolean moveCandyToFirstRow(int col) {
    if (!(box[0][col] == null)) {
        return true;
    }
    else {
    for (int i = 0; i < box.length; i++) {
        if (!(box[i][col] == null)) {
            box[0][col] = box[i][col];
            box[i][col] = null;
            return true;
        }
    }

        if (box[0][col] == null) {
            return false;
        }
    }
}
```

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.

```
public Candy removeNextByFlavor(String flavor) {

    for (int j = 0; j < box[0].length; ++) {
      for (int i = box.length-1; i >= 0; i--) {
        if (box[i][j].getFlavor.equals(flavor)) {
            Candy removal = box[i][j];
            box[i][j] = null;
            return removal;
        }
    }
    return null;
}
```

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 ●

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

```
public boolean moveCandyToFirstRow(int col)
{
    if(box[0][col] != null)
    {
      return true;
    }
    else if(box[0][col] == null)
    {
        for(int r = 1; r < box.length; r++)
        {
            if(box[r][col] == null)
            {
              return false;
            }
            else if(box[r][col] != null)
            {
                box[0][col] = box[r][col];
                box[r][col] = null;
                return true;
            }
        }
    }
}
```

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.

b)

```
Public Candy removeNextByFlavor(String flavor)
{
    for(int r=0; r< box.length; r++)
    {
        for(int c=0; c < box[0].length; c++)
        {
            if(box[r][c].getFlavor.equals(String.flavor))
            {
                box[r][c] = null;
            }
            else
            {
              return null;
            }
        }
    }
}
```

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)

```
public boolean moveCandyToFirstRow(int col){
    if(box[0][col] != null)
        return true;
    if(box[0][col] = null){
        for(int i=0; i<box.length(); i++){
            if(box[0][col+i] != null){
                box[0][col] = box[0][col+i];
                box[0][col+i] = null;
                return true;
            }
        return false;
    }
}
```

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.

b.)

```
for(int r ; r < box.length(); r++){
    for(int c; c < box[0].length(); c++){
        if(box[r][c] != null){
            String flavor = box[r][c]
        }
```

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

This question involved the manipulation of a two-dimensional (2D) array of `Candy` references and interactions between two classes. The `Candy` class included one method, `getFlavor`. A second class, `BoxOfCandy`, included an instance variable, `box`, a 2D array of `Candy` references. In addition, the class contained two methods, one to be written in part (a) and one to be written in part (b).

In part (a) students were asked to write a non-void method, `moveCandyToFirstRow`, which had one integer parameter, `col`, and needed to implement two interacting algorithms. The first algorithm moves a piece of candy, if necessary and possible, to the first row of `col`. Moving a piece of candy is deemed *necessary* if the reference to the element of `box` at row `0` column `col` is `null`, and it is *possible* if there is a `Candy` object (non-`null`) in column `col`. As a result of the second algorithm, students were expected to return `true` if there is a piece of candy in the first row of `col` upon exit—either because it was already there or because it was moved there—and `false` otherwise.

In part (b) students were asked to write a method called `removeNextByFlavor` that returns the first `Candy` object that matches the `flavor` parameter, searching the 2D array in a specified order, or returns `null` if there is not any piece of candy with the given flavor. If a piece of candy matching the flavor is found, the student was also expected to remove the reference to that object from the 2D array.

Sample: 4A
Score: 7

In part (a) point 1 was earned because each necessary value in column `col` is accessed correctly within a loop. Although starting the loop at `0` is not necessary, the response is still correct because the method returns `true` in the `if` block if a piece of candy is found at `box[0][col]`, and the `else` block and loop do not execute. If no candy is found at that location, starting the loop at `0` has no negative effect when executed. Point 2 was earned because the line `if (!(box[0][col] = null))` is a valid way to check if there is a piece of candy at row `0` and column `col`. Although the comparison should use `==`, the use of a single `=` is one of the minor errors for which no penalty is assessed on this exam. (See the "No Penalty" section on page 1 of the Scoring Guidelines

Question 4 (continued)

for a complete list.) Point 3 was earned because the response uses an `if` statement to check if there is a piece of candy at row `i`, column `col`. When the condition is `true`, the `Candy` object at row `i`, column `col` is stored at row `0`, column `col`. Then the reference to the object at row `i`, column `col` is replaced with `null`. Point 4 was earned because the appropriate `boolean` is returned in all cases. The value `true` is returned before a search is conducted if there already is a piece of candy at row `0`, column `col`. Otherwise, in the context of a loop, a search for a piece of candy (a non-empty `box` element) in column `col` is conducted. If a piece of candy is found, the correct value of `true` is returned, and the method terminates. If the entire column is searched and no candy is found, the correct value of `false` is returned. The additional `if` statement at the end of part (a) to check if there is still no candy at row `0`, column `col` is unnecessary and does not affect point 9 because `box[0][col]` must be `null` when that line is reached.

In part (b) point 5 was not earned because the traversal of the 2D array is not done as specified. Although the traversal does start in the last row, the traversal visits all elements in one column before moving on to the next column, rather than accessing all elements in one row before moving on to the next row. Point 6 was not earned because there is no check to determine if the element in `box[i][j]` is `null` before trying to access its flavor. Point 7 was earned because `getFlavor` is called on a `Candy` object. In this response, the `Candy` object is correctly accessed from `box`. However, this point could still be earned even if the `Candy` object were accessed incorrectly. Note that this point would not be earned if `getFlavor` were called on an object of a type other than `Candy` or if the call to `getFlavor` were a static call on the `Candy` class. Also, the point would not be earned if the call included any arguments. Point 8 was earned because the response compares the value returned from `getFlavor` to the parameter `flavor` using the `equals` method. A comparison using `==` would not earn this point. Point 9 was earned because the response correctly stores the value of the identified candy in the `Candy` variable `removal`. The `null` value is stored in place of the identified `Candy` object in `box` and then `removal` is returned. If the search is completed and no candy with the correct flavor is found, the response correctly returns `null`.

Sample: 4B
Score: 4

In part (a) point 1 was earned because values in column `col` are accessed correctly. The first value in the column is accessed before the loop, and the remaining values are accessed within the loop. Note that the response includes a `return` inside the loop in both the `if` and `else if` blocks of the `if` statement. As a result, the loop iterates only one time. This is assessed in point 4 and not in point 1. Point 2 was earned because the response compares the element at row `0` and column `col` of `box` to `null` using `!=`. Point 3 was earned because the response correctly uses an `if` statement to check if there is candy at row `r`, column `col`. When the `else if` condition evaluates to `true`, the value of the candy at row `r`, column `col` is stored in row `0`, column `col`. Then the reference to the object at row `r`, column `col` is replaced with `null`. Point 4 was not earned because of the early return. The value `false` should not be returned until the entire column has been searched.

Question 4 (continued)

In part (b) point 5 was not earned because the response traverses the 2D array in top-to-bottom order instead of bottom-to-top order as specified. The response includes an early return within the inner loop, but this is assessed in point 9 and not in point 5. Point 6 was not earned because the response does not check if the element at `box[r][c]` is `null` before trying to access its flavor. Point 7 was earned because `getFlavor` is called on the `Candy` object at `box[r][c]`. The missing `()` on the parameter-less method `getFlavor` is one of the minor errors for which no penalty is assessed on this exam. (See the "No Penalty" section on page 1 of the Scoring Guidelines for a complete list.) Point 8 was not earned. Although the response uses `equals` for the comparison, the argument to the `equals` method should be `flavor` instead of `String.flavor`. Point 9 was not earned for either of the following two reasons. The response identifies a candy of the given flavor and replaces the reference to the object with `null` but does not return the identified candy. Furthermore, there is an early return when the first element with a nonmatching flavor is identified.

Sample: 4C
Score: 3

In part (a) point 1 was not earned because each value in column `col` is not accessed within a loop. The loop traverses row `0`. The loop control variable `i` is used to identify the column instead of the row. Furthermore, a bounds error occurs when the `box` element in column `col + i` is accessed for values of `col + i` beyond the end of the row. The extraneous `()` on `box.length` is one of the minor errors for which no penalty is assessed on this exam. (See the "No Penalty" section on page 1 of the Scoring Guidelines for a complete list.) Point 2 was earned because the response compares the element at row `0` and column `col` of `box` to `null` using `!=`. Point 3 was not earned because the response tests and updates `box[0][col + i]` instead of `box[i][col]`. Point 4 was earned because the appropriate `boolean` is returned in all cases. Although the `Candy` object in the specific column is not correctly identified and moved to the first row, that error is assessed in point 3 and not in point 4.

In part (b) point 5 was not earned because the response traverses the 2D array in top-to-bottom order instead of bottom-to-top order as specified. Furthermore, the loop control variables `r` and `c` are not initialized. Point 6 was earned because the response checks if the element at `box[r][c]` is `null` using `!=`, and this test guards against an attempt to access the flavor of a `null` element of `box`. Point 7 was not earned because `getFlavor` is not called. Point 8 was not earned because the response does not compare a value to the parameter `flavor`. Point 9 was not earned because the response does not return a `Candy` object nor replace the referenced object with `null`.