2023

AP® Computer Science A

Scoring Guidelines

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 1: Methods and Control Structures **9 points**

Canonical solution

(a) **5 points**

```
public int findFreeBlock(int period, int duration)
{
   int blockLength = 0;

   for (int minute = 0; minute < 60; minute++)
   {
      if (isMinuteFree(period, minute))
      {
         blockLength++;
         if (blockLength == duration)
         {
            return minute - blockLength + 1;
         }
      }
      else
      {
         blockLength = 0;
      }
   }
   return -1;
}
```

(b) **4 points**

```
public boolean makeAppointment(int startPeriod,
                               int endPeriod,
                               int duration)
{
   for (int period = startPeriod;
        period <= endPeriod;
        period++)
   {
      int minute = findFreeBlock(period, duration);
      if (minute != -1)
      {
         reserveBlock(period, minute, duration);
         return true;
      }
   }
   return false;
}
```

(a) `findFreeBlock`

	Scoring Criteria	Decision Rules	
1	Loops over necessary minutes in an hour	Responses **can** still earn the point even if they • loop over fewer than 60 minutes as long as at least (60 – duration + 1) minutes are included • loop over 60 minutes and use a `boolean` to indicate that a free block has been found	**1 point**
2	Calls `isMinuteFree` with `period` and another `int` parameter	Responses **can** still earn the point even if they • call `isMinuteFree` with invalid parameters due to incorrect loop bounds Responses **will not** earn the point if they • use incorrect parameter types • order the parameters incorrectly • call the method on the class or on an object other than `this` (use of `this` is optional)	**1 point**
3	Keeps track of contiguous free minutes in a block (*algorithm*)	Responses **can** still earn the point even if they • call `isMinuteFree` incorrectly Responses **will not** earn the point if they • fail to reset when a nonfree minute is found • call `isMinuteFree` with minutes `>= 60`	**1 point**
4	Checks whether a valid block of `duration` minutes has been found	Responses **can** still earn the point even if they • maintain a `boolean` instead of accumulating the block length	**1 point**
5	Calculates and returns starting minute and `-1` appropriately based on identified block (*algorithm*)	Responses **will not** earn the point if they • are off by one on the returned value	**1 point**
		Total for part (a)	**5 points**

(b) `makeAppointment`

	Scoring Criteria	Decision Rules	
6	Loops over periods from `startPeriod` through `endPeriod` (*no bounds errors*)		**1 point**
7	Calls `findFreeBlock` and `reserveBlock` with correct number of `int` parameters, representing a period and minute as appropriate, and `duration`	Responses **can** still earn the point even if they • use incorrect parameter values Responses **will not** earn the point if they • use incorrect parameter types • order the parameters incorrectly • call the methods on the class or on an object other than `this` (use of `this` is optional)	**1 point**
8	Guards call to method to reserve a block by determining that starting minute is not `-1`		**1 point**
9	Books correct appointment and returns appropriate `boolean` (*algorithm*)	Responses **can** still earn the point even if they • have incorrect bounds in the loop • call `findFreeBlock` or `reserveBlock` incorrectly Responses **will not** earn the point if they • fail to return `true` or `false` • return before the call to `reserveBlock`	**1 point**
			4 points
	Question-specific penalties		
	None		
		Total for question 1	**9 points**

Alternate Canonical for Part (a)

```
public int findFreeBlock(int period, int duration)
{
   for (int startMin = 0; startMin < 60 - duration + 1; startMin++)
   {
      boolean isBlockFree = true;
      for (int min = 0; min < duration; min++)
      {
         if (!isMinuteFree(period, min + startMin))
         {
            isBlockFree = false;
         }
      }
      if (isBlockFree)
      {
         return startMin;
      }
   }
   return -1;
}
```

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 2: Class 9 points

Canonical solution

9 points

```
public class Sign
{
   private String message;
   private int width;

   public Sign(String m, int w)
   {
      message = m;
      width = w;
   }

   public int numberOfLines()
   {
      int len = message.length();
      if (len % width == 0)
      {
         return len / width;
      }
      else
      {
         return len / width + 1;
      }
   }

   public String getLines()
   {
      int linesNeeded = numberOfLines();
      if (linesNeeded  == 0)
      {
         return null;
      }

      String signLines = "";
      for (int i = 1; i < linesNeeded; i++)
      {
         signLines += message.substring((i - 1) * width,
                      i * width) + ";";
      }
      return signLines +
             message.substring((linesNeeded - 1) * width);
   }
}
```

`Sign`

	Scoring Criteria	Decision Rules	
1	Declares class header: `class Sign`	Responses **will not** earn the point if they • declare the class as something other than `public`	**1 point**
2	Declares appropriate `private` instance variable(s) and constructor initializes instance variables using appropriate parameters	Responses **can** still earn the point even if they • store calculated values instead of the message and width, as long as the declared instance variables can collectively answer the questions and their values are computed from the parameters (correctly or incorrectly) Responses **will not** earn the point if they • declare the variable outside the class, or in the class within a method or constructor	**1 point**
3	Declares constructor header: `Sign(String ___, int ___)`	Responses **can** still earn the point even if they • calculate values in the constructor that are returned by other methods, correctly or incorrectly, as long as the parameter types are correct Responses **will not** earn the point if they • declare the constructor as something other than `public`	**1 point**

4	Declares method headers:
	`public int numberOfLines()`
	`public String getLines()`
		Responses **will not** earn the point if they
		• omit either method
		• omit `public` or declare the method as something other than `public`
			1 point

5	`numberOfLines` divides the message length by the line width	Responses **can** still earn the point even if they • perform the division in a method other than `numberOfLines` • perform the division without using the division operator by counting line-width-sized portions of the message or by counting lines produced by the line-delimiting algorithm • incorrectly account for the final line • use a method name inconsistent with the examples, as long as it is recognizably equivalent	**1 point**
6	`numberOfLines` returns appropriate value (*algorithm*)	Responses **can** still earn the point even if they • perform the return value calculation in the constructor • return a different number of lines than `getLines` produces, as long as the number returned is the correct number for the message	**1 point**

		• return an incorrect number of lines for the message, as long as the number returned is exactly the number of lines produced by `getLines` • use a method name inconsistent with the examples, as long as it is recognizably equivalent Responses **will not** earn the point if they • incorrectly account for the final line	
7	`getLines` returns `null` appropriately	Responses **can** still earn the point even if they • identify `null` case in a method other than `getLines` • use an invalid call to `length` or `==` in guard for `null` return • use a method name inconsistent with the examples, as long as it is recognizably equivalent Responses **will not** earn the point if they • guard the return with incorrect logic	**1 point**
8	Calls `substring` and `length` (or equivalent) on `String` objects	Responses **can** still earn the point even if they • calculate `substring` parameter values incorrectly • call `substring` and/or `length` from a method other than `getLines` • use a method name inconsistent with the examples, as long as it is recognizably equivalent Responses **will not** earn the point if they • fail to call `substring` or `length` on `String` objects • call `substring` or `length` with an incorrect number of parameters, with a parameter of an incorrect type, or with incorrectly ordered parameters, anywhere in the class	**1 point**
9	`getLines` constructs the delimited sign output appropriately (*algorithm*)	Responses **can** still earn the point even if they • call `substring` and/or `length` incorrectly • fail to return the constructed `String` *(return not assessed)* • handle the empty string /`null` case incorrectly • construct the output in the constructor • use a method name inconsistent with the examples, as long as it is recognizably equivalent	**1 point**

Responses **will not** earn the point if they

- end the constructed output with a `;` or extraneous spaces
- modify the contents of `message` or `width` after they have been initialized *(no additional –1y penalty)*

Question-specific penalties
None

Total for question 2 9 points

Alternate canonical:

```
public class Sign
{
   private int numLines;
   private String lines;

   public Sign(String msg, int width)
   {
      if (!msg.equals(""))
      {
         lines = "";
         while (msg.length() > width)
         {
            lines += msg.substring(0, width) + ";";
            msg = msg.substring(width);
            numLines++;
         }
         lines += msg;
         numLines++;
      }
   }

   public int numberOfLines()
   {
      return numLines;
   }

   public String getLines()
   {
      return lines;
   }
}
```

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 3: Array / ArrayList — 9 points

Canonical solution

(a) **4 points**

```
public void cleanData(double lower, double upper)
{
   for (int i = temperatures.size() - 1; i >= 0; i--)
   {
      double temp = temperatures.get(i);
      if (temp < lower || temp > upper)
      {
         temperatures.remove(i);
      }
   }
}
```

(b) **5 points**

```
public int longestHeatWave(double threshold)
{
   int waveLength = 0;
   int maxWaveLength = 0;

   for (double temp : temperatures)
   {
      if (temp > threshold)
      {
         waveLength++;
         if (waveLength > maxWaveLength)
         {
            maxWaveLength = waveLength;
         }
      }
      else
      {
         waveLength = 0;
      }
   }
   return maxWaveLength;
}
```

(a) `cleanData`

	Scoring Criteria	Decision Rules	
1	Traverses `temperatures` (*no bounds errors*)	Responses **can** still earn the point even if they • do a forward traversal of the list • skip a value because removal from the list is handled incorrectly • use an enhanced `for` loop, as long as the list element is used in the body of the loop Responses **will not** earn the point if they • fail to ever access an element of `temperatures` in the loop • access the elements of `temperatures` incorrectly	**1 point**
2	Determines whether an element of temperature list should be removed, using `lower` and `upper`	Responses **can** still earn the point even if they • access elements of temperature list incorrectly Responses **will not** earn the point if they • apply incorrect comparison (`<` vs `<=`) or logic (`||` vs `&&`) to identify elements of list for removal	**1 point**
3	Calls `remove` (or equivalent) on temperature list with an appropriate parameter	Responses **can** still earn the point even if they • add the element to a new `ArrayList` that is not declared, is declared incorrectly, or is not assigned to the instance variable, as long as the order of elements is maintained Responses **will not** earn the point if they • call `remove` or `add` incorrectly	**1 point**
4	Removes all and only identified elements of temperature list (*algorithm*)	Responses **can** still earn the point even if they • call `remove` incorrectly • access the elements of temperature list incorrectly Responses **will not** earn the point if they • add elements to a new `ArrayList` that is not declared, is declared incorrectly, or is not assigned to the instance variable • skip a temperature list element in the traversal because removal is not handled correctly	**1 point**
		Total for part (a)	**4 points**

(b) `longestHeatWave`

	Scoring Criteria	Decision Rules	
5	Traverses `temperatures` (*no bounds errors*)	Responses **will not** earn the point if they • fail to access an element of `temperatures` in the loop • access the elements of `temperatures` incorrectly	**1 point**
6	Compares an element of temperature list to `threshold` (*in the context of a loop*)	Responses **can** still earn the point even if they • always compare `threshold` to the same list element Responses **will not** earn the point if they • apply incorrect comparison (`>` vs `>=`) to identify heat wave days	**1 point**
7	Initializes and increments the length of a heat wave (*in the context of a loop or condition*)	Responses **can** still earn the point even if they • fail to reset the length of the current heat wave when the heat wave ends	**1 point**
8	Determines the length of at least one heat wave (*algorithm*)	Responses **will not** earn the point if they • fail to reset the length of the current heat wave when the heat wave ends	**1 point**
9	Identifies the longest heat wave and returns its length (*algorithm*)		**1 point**
		Total for part (b)	**5 points**

Question-specific penalties

None

Total for question 3	**9 points**

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**