2023

AP® Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free-Response Question 2

- ☑ **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 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;
   }
}
```

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 class Sign {
    private String message;
    private int width;
    public Sign (String message, int width) {
      this.message = message;
      this.width = width;

    }
    public int numberOfLines () {
      double temp = (double)message.length() / width
      if (temp != message.length()/width) {
          return (int)temp +1;
      }else {
        return (int) temp;
    }
    public String getLines() {
        String line = ""; String temp = message
        while (temp.length() > width) {
            line += temp.substring (0, width) + ";";
            temp = temp.substring (width);
        }
        line += temp;
        return line;
    }
}
```

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 class Sign
{
    private String message;
    private int width;

    public Sign(String str, int x)
    {
        message = str;
        width = x;
    }

    public int numberOfLines()
    {
        return ((message.length() - 1) / width) + ((message.length() - 1) % width);
    }

    public String getLines()
    {
        String newMessage = "";
        int index = 0;
        for (int line = 1; line <= this.numberOfLines(); line++)
        {
            if(index >= message.length())
            {
                newMessage += (message.substring(index - width));
            }
            else
            {
                newMessage += message.substring(index, index + width) + "\n";
            }
            index += width;
        }
        return newMessage;
    }
```

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 class signs (){

 private string message;
 private int width;

 Public int numberOfLines (){
   return width;
 }
 Public string addLines (){
 string newMessage = "";
 if (message.length() % width == 0){
 for (int i= 0; i < message.length; i+=width){
  newMessage += message.substring (i, i+width) + "\n";
    }
 return newMessage.substring (0, newMessage.length()-1);
  } else {
  return null;
  }
 }
}
```

Question 2

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 define a new type by creating a class.
- 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.

Students were asked to write a class named `Sign` that would contain a constructor and two methods. In implementing a solution, students were expected to demonstrate an understanding of class header, class constructor, and method header syntax. Students were expected to properly declare and initialize private instance variables to maintain the necessary information, typically the message to be delimited and the width of the lines in the sign.

The specification of the class required that two methods be implemented: `numberOfLines` needed to return the number of lines that the sign would have based on the message and the given width and `getLines` needed to return the delimited string, inserting a `";"` to break the given message into segments of a given width without inserting the `";"` on the final string segment. The details of these methods included somewhat more algorithmic complexity than some other Class Design problems have; in this case each of the specified methods required some algorithmic work, with no simple accessor or mutator methods needed for the design.

This question also tested the student's ability to work with string values. The student needed to understand how to use the `String` methods `substring` and `length` with correct method call syntax and arguments. The question also required the student to know how to check whether a `String` variable contained the empty string and to return `null` in that case.

Sample: 2A
Score: 8

Point 1 was earned because the class header is correct. The access is `public` and specifies `class Sign` (with no parentheses). Point 2 was earned because appropriate `private` instance variables are correctly declared and then initialized in the constructor using the appropriate parameters. The use of `this.` notation before the instance variable name is correct syntax for initializing an instance variable with the same name as a parameter. Point 3 was earned because the constructor header is correct. Responses will not earn the point if the constructor access is declared as anything other than `public`. Furthermore, there cannot be a return type and the parameter order must be `String, int`. Point 4 was earned because both method headers are correct. The access must be declared `public`, there must be a correct return type, and there cannot be any method parameters. This point would not be earned if either method were omitted. Point 5 was earned because `numberOfLines` divides the message length by the line width. Point 6 was earned because `numberOfLines` properly returns the number of lines required to display the message. The response correctly tests if the message length is

Question 2 (continued)

a multiple of the line width by comparing the `double` result of division to the `int` result of division. Note that a `double` can be compared for equivalence to an `int` (e.g., `3.0 == 3` evaluates to `true`). If the message length is a multiple of the line length, the result of the division is correctly cast to `int` and returned. If not, `1` is added to account for the partial last line. Point 7 was not earned because `getLines` does not check for an empty string and return `null` appropriately. Note that the response could still earn point 7 if the `null` case were identified in a method other than `getLines`. However, the response does not include any attempt to identify the `null` case. Point 8 was earned because `getLines` calls `substring` and `length` on `String` objects using appropriate arguments. Note that a response could still earn point 8 even if `substring` and/or `length` were called from a method other than `getLines`. Point 9 was earned because the algorithm successfully delimits the message, adding semicolons to every line except for the last line. The response initializes the variable `line` to an empty string. The response uses the local variable `temp` to store a copy of the message that can be safely modified without modifying the original message stored in the instance variable. The first `width` characters of the message are repeatedly removed and appended, along with a semicolon, to `line`. The last line of the message is correctly appended, without a semicolon, after the loop.

Sample: 2B
Score: 6

Point 1 was earned because the class header is correct. The access is `public` and specifies `class Sign` (with no parentheses). Point 2 was earned because appropriate `private` instance variables are correctly declared and then initialized in the constructor using the appropriate parameters. Point 3 was earned because the constructor header is correct. Responses will not earn the point if the constructor access is declared as anything other than `public`. Furthermore, there cannot be a return type, and the parameter order must be `String`, `int`. Point 4 was earned because both method headers are correct. The access must be declared `public`, there must be a correct return type, and there cannot be any method parameters. Point 5 was earned because `numberOfLines` divides the message length by the line width. Point 6 was not earned because `numberOfLines` improperly handles the last partial line. Instead of adding 1 in the case of a partial last line, the response adds `message.length() % width`, which could result in the addition of a value greater than one. Point 7 was not earned because `getLines` does not check for an empty string and return `null` appropriately. Point 8 was earned because `getLines` calls `substring` and `length` on `String` objects using appropriate arguments. Note that a response could still earn point 8 even if `substring` and/or `length` were called from a method other than `getLines`. Point 9 was not earned because the algorithm does not properly handle the last line of the message. The condition `index >= message.length()` is always false; therefore, the `else` block is executed in each iteration of the loop, including for the last line of the message. If there is no partial last line, the code in the `else` block appends an extra semicolon at the end. If there is a partial last line, the call to `substring` in the `else` block goes past the length of the message and throws an exception.

Question 2 (continued)

Sample: 2C
Score: 2

Point 1 was not earned because the response includes parentheses in the class header. While the correct class name is `Sign` (not `Signs`), this is considered a spelling discrepancy where there is no ambiguity, which is a minor "No Penalty" error. (See the "No Penalty" section on page 1 of the Scoring Guidelines for a complete list.) Point 2 was not earned because, although appropriate `private` instance variables are declared, there is no constructor to initialize them. Point 3 was not earned because there is no constructor. Point 4 was earned because both method headers are correct. The access must be declared `public`, there must be a correct return type, and there cannot be any method parameters. Point 5 was not earned because `numberOfLines` does not divide the message length by the line width. Note that division could take place in another part of the class or could be computed algorithmically without the use of the division operator. However, because the `for` loop in `getLines` does not traverse the message, this response did not earn the point. Point 6 was not earned because `numberOfLines` returns the sign width instead of the number of lines required to display the message. Point 7 was not earned because `getLines` guards the `null` return with incorrect logic. To earn the point, a response must determine if the required number of lines is `0`. This can be done by determining if `message` is an empty string. Instead, this response checks if the length of `message` is evenly divisible by `width`. Point 8 was earned because `getLines` calls `substring` and `length` on `String` objects using appropriate arguments. Note that a response could still earn point 8 even if `substring` and/or `length` were called from a method other than `getLines`. Point 8 could also be earned with incorrect arguments to `substring`; this is assessed in point 9. Point 9 was not earned because the algorithm does not delimit the message properly. Even if the `for` loop had used `i += width` to update the loop variable correctly, the call to `substring` inside the loop would go out-of-bounds when accessing the last partial line.