2021

AP CollegeBoard

AP® Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free Response Question 2

- ☑ **Scoring Guideline**
- ☑ **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 Design — 9 points

Canonical solution

9 points

```
public class CombinedTable
{
   private SingleTable table1;
   private SingleTable table2;

   public CombinedTable(SingleTable tab1, SingleTable tab2)
   {
      table1 = tab1;
      table2 = tab2;
   }

   public boolean canSeat(int n)
   {
      if (table1.getNumSeats() + table2.getNumSeats() - 2 >= n)
      {
         return true;
      }
      else
      {
         return false;
      }
   }

   public double getDesirability()
   {
      if (table1.getHeight() == table2.getHeight())
      {
         return (table1.getViewQuality() +
                 table2.getViewQuality()) / 2;
      }
      else
      {
         return ((table1.getViewQuality() +
                  table2.getViewQuality()) / 2) - 10;
      }
   }
}
```

`CombinedTable`

	Scoring Criteria	Decision Rules	

1	Declares class header:
	`class CombinedTable`
	and constructor header:
	`CombinedTable(SingleTable ___, SingleTable ___)`
	(*must not be* `private`)
		Responses **can** still earn the point even if they declare the class header as `class CombinedTable extends SingleTable`	**1 point**

2	Declares appropriate `private` instance variables including at least two `SingleTable` references	Responses **can** still earn the point even if they declare an additional instance variable to cache the number of seats at the combined table Responses **will not** earn the point if they • declare and initialize local variables in the constructor instead of instance variables • declare additional instance variable(s) that cache the desirability rating • omit keyword `private` • declare variables outside the class	**1 point**
3	Constructor initializes instance variables using parameters	Responses **can** still earn the point even if they declare and initialize local variables in the constructor instead of instance variables	**1 point**
4	Declares header: `public boolean canSeat(int ___)`		**1 point**
5	Calls `getNumSeats` on a `SingleTable` object	Responses **can** still earn the point even if they call `getNumSeats` on constructor parameters or local variables of type `SingleTable` in the constructor Responses **will not** earn the point if they call the `SingleTable` accessor method on something other than a `SingleTable` object	**1 point**
6	`canSeat(n)` returns `true` if and only if sum of seats of two tables `- 2 >= n`	Responses **can** still earn the point even if they call `getNumSeats` incorrectly	**1 point**
7	Declares header: `public double getDesirability()`		**1 point**
8	Calls `getHeight` and `getViewQuality` on `SingleTable` objects	Responses **can** still earn the point even if they call `getHeight` or `getViewQuality` on constructor parameters or local variables of type `SingleTable` in the constructor	**1 point**

		Responses **will not** earn the point if they call the `SingleTable` accessor methods on something other than a `SingleTable` object	
9	`getDesirability` computes average of constituent tables' view desirabilities	Responses **can** still earn the point even if they • call `getHeight` or `getViewQuality` on constructor parameters or local variables of type `SingleTable` in the constructor • fail to return the computed average (*return is not assessed*) Responses **will not** earn the point if they • fail to have an `if` statement and a correct calculation • choose the incorrect value (average vs. average – 10) based on evaluation of the `if` statement condition	**1 point**

Question-specific penalties
None

Total for question 2 9 points

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

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

```
public class CombinedTable
{   private SingleTable a;
    private SingleTable b;

    public CombinedTable (SingleTable tablea, SingleTable tableb)
    {   a = tablea;
        b = tableb;
    }
    public boolean canSeat (int num)
    {   if (num <= (a.getNumSeats() + b.getNumSeats()-2))
            {return true;}
        return false;
    }
    public double getDesirability()
    {   if(a.getHeight() == b.getHeight())
        { return(a.getViewQuality() + b.getViewQuality())/2;
        }
        else
        { return (a.getViewQuality() + b.getViewQuality)/2-10;
        }
    }
```

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

```
public class CombinedTable {
 private int seats;
 private double desirability;

 public CombinedTable(SingleTable t1, SingleTable t2) {
seats = t1.getNumSeats() + t2.getNumSeats() - 2;
if (t1.getHeight() == t2.getHeight()){
  desirability = (t1.getViewQuality + t2.getViewQuality)/2;
  }
else {
 desirability = ((t1.getViewQuality + t2.getViewQuality)/2) - 10;
  }

 }
public void getDesirability() {
return desirability;
 }
public boolean canSeat(int num) {
 if(seats == num) {
  return true;
 else {
 return false;
  }
}
```

Q2 Sample C 1 of 1

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

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

```
public class CombinedTable {
  CombinedTable c1 = new CombinedTable (t1, t2);

      public CombinedTable (t1, t2) {

      }

      public boolean canSeat (num) {
       if ( num <= 10) {
            return true;
            }
          return false;
       }

    public double getDesirability ( double des) {
     IF (t1.getHeight() == t2.getHeight()) {
        des = (t1.getViewQuality + t2.getViewQuality)/2.0;
        }
      else {
        des = (t1.getViewQuality + t2.getViewQuality)/2.0) + 10;
        }
      return des;
    }
}
```

Question 2

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 methods using expressions and conditional statements.

Students were asked to design the class `CombinedTable`, which represents a table composed of two single tables pushed together. The students were given a partial definition of the class `SingleTable`, which represents a table at a restaurant, to be used in their `CombinedTable` class design. Students were expected to demonstrate an understanding of class constructor and method header syntax. Additionally, students were expected to determine the data types and instance variables needed to track the information shown in the example. Students were then expected to correctly declare, initialize, access, and generate the appropriate values from their data members. Students were expected to properly protect the data members by declaring them as `private` and properly define the methods `canSeat` and `getDesirability`. Students had to recognize that they could not compute and store the desirability value in the constructor because the design of `SingleTable` allowed for a table view quality to change at any time; `getDesirability` always had to reflect the latest values of the `SingleTable` view quality.

Sample: 2A
Score: 9

Point 1 was earned because the response correctly declares the class header and constructor header. The response correctly names the constructor and provides two `SingleTable` parameters. Both the class and the constructor are declared as `public`. Point 2 was earned because the response declares two `SingleTable` instance variables. The view quality of a constituent table may change after a `CombinedTable` is constructed. The point verifies that the design of the response supports this by requiring `SingleTable` instance variables. The instance variables must be declared as `private`. Point 3 was earned because the constructor initializes the instance variables with parameter values. The point tests not only the assignment of a parameter to an instance (or local) variable but also that the assignment involves consistent types. Point 3 does not assess the usefulness of the values assigned. Point 4 was earned because the response correctly declares the header for the `canSeat` method. The method must return a `boolean`, take a single `int` parameter, and have a `public` access specifier. Point 5 was earned because the response calls the `getNumSeats` method on `SingleTable` objects. The response calls the method correctly with instance variables `a` and `b`. In the response, the call to `getNumSeats` occurs in the `canSeat` method. Point 6 was earned because the response chooses the correct return value, based on the correct comparison, with a correct seat calculation. The point evaluates the logic of the `canSeat` method. To earn the point, both the calculation of available seats and the returned value must be correct. Point 7 was earned because the header for the `getDesirability` method is correct. To earn the point, the method must be declared as `public`, must have a return type of `double`, and must have an empty parameter list. Point 8 was earned because the response correctly calls both the `getHeight` and `getViewQuality` methods on `SingleTable` objects. The response calls both methods on `SingleTable` instance variables in the `getDesirability` method. The missing `()` on the parameter-less method invocation (on the fourth call to `getViewQuality`) is one of the minor errors for which no penalty is assessed. (The "No Penalty" category on page 1 of the Scoring Guidelines contains a complete list of these errors.) Point 9 was earned because the response correctly computes the average desirability of the `CombinedTable`. The point evaluates the calculation only. To earn the point, a response must compare the heights of the `SingleTable` objects, choose the correct formula, and calculate the desirability accordingly.

Question 2 (continued)

Sample: 2B
Score: 6

Point 1 was earned because the response correctly declares the class and the constructor has the correct number and type of parameters (two `SingleTable` objects), is `public`, and is correctly named. Point 2 was not earned because the response does not declare any `SingleTable` instance variables. The response is a relatively clear example of a common solution strategy that involves performing seat and desirability computations immediately, in the constructor, and caching the results in instance variables. Although the desirability computation may be done correctly (earning other points), the view quality of constituent tables may later change, and if the stored desirability is never updated, it will then be incorrect. A design for the class cannot be correct without storing `SingleTable` references to both constructor parameters. (This design choice is *only* assessed in point 2.) Point 3 was earned because the response initializes its instance variables using parameter values. In this case, the first instance variable is an `int` (`seats`) and is assigned an integer value derived from parameters `t1` and `t2`; the second instance variable is a `double` and is also assigned a number derived from `t1` and `t2`. Point 4 was earned because the header for the `canSeat` method is correct. Point 5 was earned because the response calls the `getNumSeats` method on `SingleTable` objects `t1` and `t2`. The response makes the call in the constructor, rather than the `canSeat` method. Point 6 was not earned because the logical test is incorrect. The response compares `seats`, an instance variable correctly initialized in the constructor, to `num`, but the comparison is for equality. The response is closer than it looks to a correct solution, based on relevant initializations in the constructor. In the response, changing the logical expression from `seats == num` to `seats <= num` would provide a logically correct solution. Point 7 was not earned because the method has an incorrect return type. Point 8 was earned because the response correctly calls the `getViewQuality` and `getHeight` methods on `SingleTable` objects in the constructor. The `SingleTable` objects are parameters rather than instance variables, but this is acceptable for this point. (The missing `()` are a minor "No Penalty" error.) Point 9 was earned because desirability is correctly calculated in the constructor and stored in an instance variable.

Sample: 2C
Score: 1

Point 1 was not earned because the constructor header does not contain two `SingleTable` parameters. Point 2 was not earned because the response does not declare any `SingleTable` instance variables. Point 3 was not earned because the constructor does not initialize any instance variables with its parameters. Point 4 was not earned because the header for the `canSeat` method is missing the type of its parameter. Point 5 was not earned because the response does not call the `getNumSeats` method on a `SingleTable` object. Point 6 was not earned because the condition is incorrect. The response compares `num` with the literal integer `10` instead of the number of available seats. Point 7 was not earned because the response includes a parameter in the `getDesirability` method header declaration. Point 8 was earned because the response calls both the `getHeight` and `getViewQuality` methods on `SingleTable` objects. The variables `t1` and `t2` are not declared as `SingleTable` parameters in the constructor (thus not earning point 1) nor are they declared as `SingleTable` instance variables (thus not earning point 2). However, `t1` and `t2` can be considered as `SingleTable` values for the purpose of assessing the point because the prompt indicates the constructor parameters are `SingleTable` variables, the response uses them as constructor parameters, and the response uses them elsewhere as `SingleTable` variables. Point 9 was not earned because the response adds 10 to the view quality when it should subtract 10.