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

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.*