Question 1: Methods and Control Structures — 9 points

Canonical solution

(a) — **5 points**

```
public int scoreGuess(String guess)
{
   int count = 0;

   for (int i = 0; i <= secret.length() - guess.length(); i++)
   {
      if (secret.substring(i, i + guess.length()).equals(guess))
      {
         count++;
      }
   }

   return count * guess.length() * guess.length();
}
```

(b) — **4 points**

```
public String findBetterGuess(String guess1, String guess2)
{
   if (scoreGuess(guess1) > scoreGuess(guess2))
   {
      return guess1;
   }
   if (scoreGuess(guess2) > scoreGuess(guess1))
   {
      return guess2;
   }
   if (guess1.compareTo(guess2) > 0)
   {
      return guess1;
   }
   return guess2;
}
```

(a) `scoreGuess`

	Scoring Criteria	Decision Rules	
1	Compares `guess` to a substring of `secret`	Responses **can** still earn the point even if they only call `secret.indexOf(guess)` Responses **will not** earn the point if they use `==` instead of `equals`	**1 point**
2	Uses a substring of `secret` with correct length for comparison with `guess`	Responses **can** still earn the point even if they • only call `secret.indexOf(guess)` • use `==` instead of `equals`	**1 point**
3	Loops through all necessary substrings of `secret` (*no bounds errors*)	Responses **will not** earn the point if they skip overlapping occurrences	**1 point**
4	Counts number of identified occurrences of `guess` within `secret` (*in the context of a condition involving both* `secret` *and* `guess`)	Responses **can** still earn the point even if they • initialize count incorrectly or not at all • identify occurrences incorrectly	**1 point**
5	Calculates and returns correct final score (*algorithm*)	Responses **will not** earn the point if they • initialize count incorrectly or not at all • fail to use a loop • fail to compare `guess` to multiple substrings of `secret` • count the same matching substring more than once • use a changed or incorrect `guess` length when computing the score	**1 point**
		Total for part (a)	**5 points**

(b) `findBetterGuess`

	Scoring Criteria	Decision Rules	
6	Calls `scoreGuess` to get scores for `guess1` and `guess2`	Responses **will not** earn the point if they • fail to include parameters in the method calls • call the method on an object or class other than `this`	**1 point**
7	Compares the scores	Responses **will not** earn the point if they • only compare using `==` or `!=` • fail to use the result of the comparison in a conditional statement	**1 point**
8	Determines which of `guess1` and `guess2` is alphabetically greater	Responses **can** still earn the point even if they reverse the comparison Responses **will not** earn the point if they • reimplement `compareTo` incorrectly • use result of `compareTo` as if `boolean`	**1 point**
9	Returns the identified `guess1` or `guess2` (*algorithm*)	Responses **can** still earn the point even if they • call `scoreGuess` incorrectly • compare strings incorrectly Responses **will not** earn the point if they • reverse a comparison • omit either comparison • fail to return a guess in some case	**1 point**
		Total for part (b)	**4 points**
	Question-specific penalties		
	None		
		Total for question 1	**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.*