2025

# 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.*

## 2025 Digital Decision Rules

- Some non-ASCII characters are not printing correctly in ONE, particularly extended punctuation. Certain kinds of double-quotes may display as â□□; a character that displays as ï¼□ may be a parenthesis, comma, or semicolon (or other punctuation). If the badly displayed character makes sense as one of those, evaluate the response accordingly.
- If there are missing closed-double-quotes or closed-parentheses, assume they are at the end of the line where they opened, immediately before the semicolon or curly bracket (if any).
- Assume an open/left curly bracket `{` immediately after any method header or class header that does not already have one.
- Assume an appropriate amount of closing brackets before any method header to close all open brackets from the previous method or constructor.
- If there are missing curly brackets, clear indentation can "convey intent". Evaluate the response accordingly.
- Inside a method with left-justified code, indentation cannot "convey intent", so missing curly brackets cannot be assumed. Without bracketing or indentation, only the first line of a `while` / `if` / `for` is controlled by the statement; with open curly bracket and no indentation cues, the entire remainder of the method is "inside" the statement.

**No Penalty**

- `:` instead of `;` and vice versa
- `,` instead of `;` and vice versa

## Question 1: Methods and Control Structures — 9 points

**Canonical solution**

**a.** **4 points**

```
public int walkDogs(int hour)
{
   int dogsToWalk = company.numAvailableDogs(hour);

   if (dogsToWalk > maxDogs)
   {
      dogsToWalk = maxDogs;
   }

   company.updateDogs(hour, dogsToWalk);
   return dogsToWalk;
}
```

**b.** **5 points**

```
public int dogWalkShift(int startHour, int endHour)
{
   int totalPay = 0;

   for (int hour = startHour; hour <= endHour; hour++)
   {
      int dogs = walkDogs(hour);
      int hourPay = 5 * dogs;
      if (dogs == maxDogs || (hour >= 9 && hour <= 17))
      {
         hourPay += 3;
      }
      totalPay += hourPay;
   }
   return totalPay;
}
```

**a.** `walkDogs`

| | Scoring Criteria | Decision Rules | |
|---|---|---|---|
| **1** | Calls `DogWalkCompany` method(s) on `company` | Responses **can** still earn the point even if they<br>• fail to save or use the returned value<br>• call `numAvailableDogs` more than once<br>• only call one of the methods<br><br>Responses **will not** earn the point if they<br>• use the incorrect parameter types<br>• call either `numAvailableDogs` or `updateDogs` on nothing or on something other than `company` | **1 point** |
| **2** | Compares available dogs and `maxDogs` to determine number of dogs to walk | Responses **can** still earn the point even if they<br>• call `numAvailableDogs` incorrectly<br>• calculate an incorrect number of dogs that were walked | **1 point** |
| **3** | Calls `numAvailableDogs` with `hour` and `updateDogs` with `hour` and correct number of dogs to walk (*algorithm*) | Responses **can** still earn the point even if they<br>• call either method on nothing or on something other than `company`<br><br>Responses **will not** earn the point if they<br>• calculate an incorrect number of dogs that were walked | **1 point** |
| **4** | Returns calculated value | Responses **can** still earn the point even if they<br>• calculate an incorrect number of dogs that were walked<br>• call `numAvailableDogs` multiple times<br><br>Responses **will not** earn the point if they<br>• return something other than an `int`<br>• fail to return a value in some cases<br>• print a value in addition to or instead of returning it | **1 point** |

**b.** `dogWalkShift`

| | Scoring Criteria | Decision Rules | |
|---|---|---|---|
| **5** | Loops from `startHour` to `endHour`, inclusive | Responses **can** still earn the point even if they<br>• return from the loop early, as long as bounds would otherwise be correct | **1 point** |
| **6** | Calls `walkDogs` with `int` parameter (*in the context of a loop*) | Responses **can** still earn the point even if they<br>• fail to save or use the returned value<br><br>Responses **will not** earn the point if they<br>• use an incorrect parameter type on any call to `walkDogs`<br>• call the method on the class or on an object other than `this` (use of `this` is optional) | **1 point** |
| **7** | Calculates base pay for an hour | Responses **can** still earn the point even if they<br>• fail to include the calculation in the loop<br>• call `walkDogs` incorrectly | **1 point** |
| **8** | Calculates possible bonus for an hour's pay | Responses **can** still earn the point even if they<br>• call `walkDogs` multiple times inside the loop<br>• fail to include the calculation in the loop<br><br>Responses **will not** earn the point if they<br>• count the bonus twice when both conditions are true<br>• count the bonus only when both conditions are true<br>• count the bonus when it has not been earned<br>• calculate bonus incorrectly | **1 point** |
| **9** | Accumulates total pay (*algorithm*) | Responses **can** still earn the point even if they<br>• calculate base pay or bonus incorrectly<br>• fail to return total pay (*return not assessed in this part*)<br><br>Responses **will not** earn the point if they<br>• fail to initialize variable used for total pay<br>• call `walkDogs` multiple times inside the loop<br>• do not accumulate base and bonus in the context of a loop<br>• return from the loop early | **1 point** |
| | **Question-specific penalties** | | |
| | None | | |

## 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.*

## 2025 Digital Decision Rules

- Some non-ASCII characters are not printing correctly in ONE, particularly extended punctuation. Certain kinds of double-quotes may display as â□□; a character that displays as ï¼□ may be a parenthesis, comma, or semicolon (or other punctuation). If the badly displayed character makes sense as one of those, evaluate the response accordingly.
- If there are missing closed-double-quotes or closed-parentheses, assume they are at the end of the line where they opened, immediately before the semicolon or curly bracket (if any).
- Assume an open/left curly bracket `{` immediately after any method header or class header that does not already have one.
- Assume an appropriate amount of closing brackets before any method header to close all open brackets from the previous method or constructor.
- If there are missing curly brackets, clear indentation can "convey intent". Evaluate the response accordingly.
- Inside a method with left-justified code, indentation cannot "convey intent", so missing curly brackets cannot be assumed. Without bracketing or indentation, only the first line of a `while` / `if` / `for` is controlled by the statement; with open curly bracket and no indentation cues, the entire remainder of the method is "inside" the statement.

**No Penalty**

- `:` instead of `;` and vice versa
- `,` instead of `;` and vice versa

## Question 2: Class — 9 points

**Canonical solution**

**9 points**

```
public class SignedText
{
   private String firstName;
   private String lastName;

   public SignedText(String first, String last)
   {
      firstName = first;
      lastName = last;
   }

   public String getSignature()
   {
      String sig = "";
      if (!firstName.equals(""))
      {
         sig += firstName.substring(0, 1) + "-";
      }
      sig += lastName;
      return sig;
   }

   public String addSignature(String textStr)
   {
      String sig = getSignature();
      int index = textStr.indexOf(sig);

      if (index == -1)
      {
         return textStr + sig;
      }
      else if (index == 0)
      {
         return textStr.substring(sig.length()) + sig;
      }
      else
      {
         return textStr;
      }
   }
}
```

`SignedText`

| | Scoring Criteria | Decision Rules | |
|---|---|---|---|
| **1** | Declares class header:<br>`class SignedText` | Responses **will not** earn the point if they<br>• declare the class as `private`<br>• include extraneous code outside the class<br>• include `()` with or without parameters in the class header | **1 point** |
| **2** | Declares all appropriate `private` instance variable(s) and constructor initializes instance variable(s) using appropriate parameters | Responses **can** still earn the point even if they<br>• create and store the signature using only one `String` instance variable<br><br>Responses **will not** earn the point if they<br>• declare any instance variables `static`<br>• omit `private` in instance variable declaration<br>• declare variables outside the class<br>• declare an instance variable inside the constructor<br>• fail to use either parameter<br>• assign values to instance variables from an unknown source<br>• assign initial value to local variable instead of instance variable, even if the local variables are declared as `private`<br>• assign parameter(s) to variable(s) with incompatible data type<br>• return a value from the constructor<br>• define the constructor inside a method or define a method inside the constructor | **1 point** |
| **3** | Declares constructor header:<br>`SignedText(String ___,`<br>`String ___)` | Responses **will not** earn the point if they<br>• declare the constructor as `private` or `static` | **1 point** |
| **4** | Declares method headers:<br>`public String`<br>`getSignature()`<br>`public String`<br>`addSignature(String ___)` | Responses **will not** earn the point if they<br>• omit `public` in either method header or declare either method as something other than `public`<br>• use incorrect method name<br>• use incorrect return or parameter type | **1 point** |
| **5** | Compares first name to the empty string | Responses **can** still earn the point even if they<br>• perform the comparison implicitly (e.g., by comparing the string's length to 0) | **1 point** |

| | | | |
|---|---|---|---|
| **6** | Determines appropriate signature string in both cases (*algorithm*) | Responses **can** still earn the point even if they<br>• fail to return a value in some cases (*return from* `getSignature` *not assessed*)<br><br>Responses **will not** earn the point if they<br>• construct the correct string but return something else<br>• define another method inside the signature method or vice versa | **1 point** |
| **7** | Calls `String` methods using correct syntax throughout the class | Responses **can** still earn the point even if they<br>• call `substring` only one time<br>• call `indexOf`, `contains`, `charAt`, or other appropriate methods<br><br>Responses **will not** earn the point if they<br>• only call `length` and/or `equals` without using other `String` methods<br>• fail to call at least one `String` method which accesses elements of a string | **1 point** |
| **8** | Identifies the three required cases for `addSignature` using appropriate conditions | Responses **can** still earn the point even if they<br>• return an incorrect string in one or more cases | **1 point** |
| **9** | `addSignature` returns appropriate `String` in all three cases (*algorithm*) | Responses **can** still earn the point even if they<br>• identify one or more of the three cases incorrectly, as long as they are unambiguously no-match, match-start, and match-end<br>• implement `getSignature` incorrectly<br><br>Responses **will not** earn the point if they<br>• call `getSignature` incorrectly, or incorrectly implement its functionality in `addSignature`<br>• fail to identify three distinct cases<br>• build correct strings but assign them to cases incorrectly<br>• print the string instead of or in addition to returning it<br>• define another method inside `addSignature` or vice versa | **1 point** |

| | **Question-specific penalties** |
|---|---|
| | None |

**Alternate canonical:**

```
public class SignedText
{
   private String signature;

   public SignedText(String first, String last)
   {
      signature = last;
      if (first.length() > 0)
      {
         signature = first.substring(0, 1) + "-" + last;
      }
   }

   public String getSignature()
   {
      return signature;
   }

   public String addSignature(String textStr)
   {
      String result = textStr;
      int index = textStr.indexOf(signature);
      if (index < 0)
      {
         result += signature;
      }
      else if (index == 0)
      {
         result = result.substring(signature.length()) + signature;
      }
      return result;
   }
}
```

## 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.*

## 2025 Digital Decision Rules

- Some non-ASCII characters are not printing correctly in ONE, particularly extended punctuation. Certain kinds of double-quotes may display as â□□; a character that displays as ï¼□ may be a parenthesis, comma, or semicolon (or other punctuation). If the badly displayed character makes sense as one of those, evaluate the response accordingly.
- If there are missing closed-double-quotes or closed-parentheses, assume they are at the end of the line where they opened, immediately before the semicolon or curly bracket (if any).
- Assume an open/left curly bracket `{` immediately after any method header or class header that does not already have one.
- Assume an appropriate amount of closing brackets before any method header to close all open brackets from the previous method or constructor.
- If there are missing curly brackets, clear indentation can "convey intent". Evaluate the response accordingly.
- Inside a method with left-justified code, indentation cannot "convey intent", so missing curly brackets cannot be assumed. Without bracketing or indentation, only the first line of a `while` / `if` / `for` is controlled by the statement; with open curly bracket and no indentation cues, the entire remainder of the method is "inside" the statement.

**No Penalty**

- `:` instead of `;` and vice versa
- `,` instead of `;` and vice versa

## Question 3: Array / ArrayList — 9 points

**Canonical solution**

**a.** **4 points**

```
public Round(String[] names)
{
   competitorList = new ArrayList<Competitor>();
   for(int i = 0; i < names.length; i++)
   {
      Competitor c = new Competitor(names[i], i + 1);
      competitorList.add(c);
   }
}
```

**b.** **5 points**

```
public ArrayList<Match> buildMatches()
{
   ArrayList<Match> matches = new ArrayList<Match>();

   if (competitorList.size() % 2 == 0)
   {
      for(int i = 0; i < competitorList.size()/2; i++)
      {
         matches.add(new Match(competitorList.get(i),
               competitorList.get(competitorList.size() – i - 1)));
      }
   }
   else
   {
      for(int i = 1; i < competitorList.size() / 2 + 1; i++)
      {
         matches.add(new Match(competitorList.get(i),
             competitorList.get(competitorList.size() - i)));
      }
   }
   return matches;
}
```

**a.** `Round`

| | Scoring Criteria | Decision Rules | |
|---|---|---|---|
| **1** | Accesses* all elements of `names` (*no bounds errors*) | Responses **will not** earn the point if they<br>• access `names` incorrectly | **1 point** |
| **2** | Initializes and maintains rank associated with each accessed competitor, beginning at 1 | Responses **can** still earn the point even if they<br>• maintain a rank variable with initial value 0, as long as it is offset by 1 when accessed<br>• fail to construct a `Competitor` object<br><br>Responses **will not** earn the point if they<br>• compute rank incorrectly for any competitor | **1 point** |
| **3** | Constructs `Competitor` with provided name and computed rank | Responses **can** still earn the point even if they<br>• access names incorrectly<br>• compute rank incorrectly<br><br>Responses **will not** earn the point if they<br>• omit `new` in the construction of a `Competitor` or fail to use the correct number and type of parameters | **1 point** |
| **4** | Adds all constructed competitors to `competitorList` in the correct order (*algorithm*) | Responses **can** still earn the point even if they<br>• fail to initialize `competitorList` *(`ArrayList` creation not assessed in this part)*<br>• omit `new` in the construction of a `Competitor`<br>• fail to access all elements of `names`<br>• add elements with an incorrect or missing rank<br><br>Responses **will not** earn the point if they<br>• add items without constructing a `Competitor`<br>• access `competitorList` incorrectly<br>• fail to update the instance variable `competitorList` (e.g. by redeclaring as a local variable)<br>• print or return a value instead of or in addition to updating `competitorList` | **1 point** |

*An enhanced `for` loop inherently accesses all elements of an `ArrayList`

**b.** `buildMatches`

| | Scoring Criteria | Decision Rules | |
|---|---|---|---|
| **5** | Declares and initializes local `ArrayList` of `Match` objects | Responses **will not** earn the point if they<br>• fail to declare the `ArrayList`, even if they have other local variables declared | **1 point** |
| **6** | Initializes and maintains an index used to move from both ends of `competitorList` *(algorithm)* | Responses **can** still earn the point even if they<br>• start at 0 instead of 1 or vice versa<br>• make a bounds error with the "high" index, as long as it is counting down from the end of the list<br>• maintain two separate index variables instead of a single offset or other equivalent strategy, as long as it is used to count up from the start and down from the end<br>• fail to use the indices to construct a `Match` | **1 point** |
| **7** | Computes starting low index based on even/odd comparison | Responses **can** still earn the point even if they<br>• compute the index incorrectly, as long as the even/odd cases start at different indices<br>• call the `size` method incorrectly<br>• modifies `competitorList` instead of skipping an index<br><br>Responses **will not** earn the point if they<br>• determine even and odd number of competitors incorrectly | **1 point** |
| **8** | Gets two elements of `competitorList` based on maintained index, constructs a `Match` object from them, and adds it to the local list | Responses **can** still earn the point even if they<br>• omit `new` in the creation of the `Match` object *(use of `new` not assessed for this type)*<br>• use incorrect indices<br><br>Responses **will not** earn the point if they<br>• access `competitorList` incorrectly<br>• fail to create an object of type `Match`<br>• use incorrect number or type of parameters on any of the method calls | **1 point** |

| | | | |
|---|---|---|---|
| **9** | Populates the list with the correct number of pairs of competitors, omitting the first competitor in the odd case, without bounds errors *(algorithm)* | Responses **can** still earn the point even if they<br>• fail to return the `ArrayList` (*return not assessed in this question*)<br>• determine even and odd number of competitors incorrectly, as long as the cases are unambiguous<br>• fail to create or use a `Match` object<br><br>Responses **will not** earn the point if they<br>• fail to guard against even and odd numbers of competitors<br>• include the first competitor for odd numbers of competitors or omit the first for even numbers<br>• include too many matches (e.g. due to continuing past the middle of the list)<br>• add indices instead of competitors<br>• make syntactically incorrect call(s) to `size` or other `ArrayList` methods<br>• permanently modify `competitorList` | **1 point** |
| | **Question-specific penalties** | | |
| | None | | |

Alternate canonical solution:

```
public Round(String[] names)
{
    competitorList = new ArrayList<Competitor>();

    int rank = 1;

    for (String name : names)
    {
       Competitor c = new Competitor(name, rank);
       competitorList.add(c);
       rank++;
    }
}


public ArrayList<Match> buildMatches()
{
   ArrayList<Match> matches = new ArrayList<Match>();

   int low = 0;
   if (competitorList.size() % 2 == 1)
   {
      low = 1;
   }
   int high = competitorList.size() - 1;

   while (low < high)
   {
      Match m = new Match(competitorList.get(low),
                          competitorList.get(high));
      matches.add(m);
      low++;
      high--;
   }
   return matches;

}
```

## 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.*

## 2025 Digital Decision Rules

- Some non-ASCII characters are not printing correctly in ONE, particularly extended punctuation. Certain kinds of double-quotes may display as â□□; a character that displays as ï¼□ may be a parenthesis, comma, or semicolon (or other punctuation). If the badly displayed character makes sense as one of those, evaluate the response accordingly.
- If there are missing closed-double-quotes or closed-parentheses, assume they are at the end of the line where they opened, immediately before the semicolon or curly bracket (if any).
- Assume an open/left curly bracket `{` immediately after any method header or class header that does not already have one.
- Assume an appropriate amount of closing brackets before any method header to close all open brackets from the previous method or constructor.
- If there are missing curly brackets, clear indentation can "convey intent". Evaluate the response accordingly.
- Inside a method with left-justified code, indentation cannot "convey intent", so missing curly brackets cannot be assumed. Without bracketing or indentation, only the first line of a `while` / `if` / `for` is controlled by the statement; with open curly bracket and no indentation cues, the entire remainder of the method is "inside" the statement.

**No Penalty**

- `:` instead of `;` and vice versa
- `,` instead of `;` and vice versa

## Question 4: 2D Arrays — 9 points

**Canonical solution**

**a.** **4 points**

```
public SumOrSameGame(int numRows, int numCols)
{
   puzzle = new int[numRows][numCols];

   for (int row = 0; row < numRows; row++)
   {
      for (int col = 0; col < numCols; col++)
      {
         puzzle[row][col] = (int)(Math.random() * 9) + 1;
      }
   }
}
```

**b.** **5 points**

```
public boolean clearPair(int row, int col)
{
   int val1 = puzzle[row][col];
   for (int currRow = row; currRow < puzzle.length; currRow++)
   {
      for (int currCol = 0; currCol < puzzle[0].length; currCol++)
      {
         int val2 = puzzle[currRow][currCol];
         if (currRow != row || currCol != col)
         {
            if (val1 == val2 || val1 + val2 == 10)
            {
               puzzle[row][col] = 0;
               puzzle[currRow][currCol] = 0;
               return true;
            }
         }
      }
   }
   return false;
}
```

**a.** `SumOrSameGame`

| | Scoring Criteria | Decision Rules | |
|---|---|---|---|
| **1** | Constructs correctly sized 2D array of `int` and assigns to instance variable `puzzle` | Responses **will not** earn the point if they<br>• fail to update the instance variable `puzzle`, e.g., by redeclaring as a local variable<br>• print or return a value instead of or in addition to updating `puzzle` | **1 point** |
| **2** | Traverses 2D array (*no bounds errors*) | Responses **can** still earn the point even if they<br>• fail to construct the 2D array, as long as traversal uses correct bounds<br>• fail to assign to the 2D array elements<br><br>Responses **will not** earn the point if they<br>• fail to access an element of the 2D array in the traversal | **1 point** |
| **3** | Generates a random integer that is uniform in the range `[1, 9]` | Responses **will not** earn the point if they<br>• fail to call `Math.random` or an equivalent method<br>• make any incorrect call to `Math.random` or an equivalent method<br>• call `random` without `Math.`<br>• fail to cast to an `int` | **1 point** |
| **4** | Assigns values to 2D array elements (*algorithm*) | Responses **can** still earn the point even if they<br>• generate the random value incorrectly<br>• assign to a local 2D array instead of the instance variable<br>• fail to assign some elements due to a bounds error<br><br>Responses **will not** earn the point if they<br>• assign a value other than the attempted randomly generated value<br>• correctly initialized the 2D array but now assign somewhere else<br>• assign the same value to all visited elements<br>• fail to assign to all visited elements | **1 point** |

**b.** `clearPair`

| | Scoring Criteria | Decision Rules | |
|---|---|---|---|
| **5** | Accesses all and only necessary elements of `puzzle` in rows `>=` `row` (*no bounds errors*) | Responses **can** still earn the point even if they<br>• access additional rows, as long as they also guard against pairing them<br>• pair the element at `row` and `col` with itself<br>• omit extra elements due to incorrect self-pairing guard, as long as the bounds would otherwise support accessing all necessary elements<br>• return early, as long as bounds and indices would otherwise support accessing all necessary elements<br><br>Responses **will not** earn the point if they<br>• fail to access elements of `puzzle` correctly | **1 point** |
| **6** | Guards against pairing an element with itself | Responses **will** not earn the point if they<br>• omit extra potential pairs with their self-pairing guard<br>• have no loop | **1 point** |
| **7** | Determines whether two elements are the same or sum to 10 | Responses **can** still earn the point even if they<br>• pair the element at `row` and `col` with itself<br>• have no loop<br><br>Responses **will not** earn the point if they<br>• test only one of the conditions | **1 point** |
| **8** | Sets a 2D array element to `0` *(in the context of a loop)* | Responses **can** still earn the point even if they<br>• set only one identified element to `0`<br>• incorrectly identify elements<br>• set more than 2 elements to `0` | **1 point** |
| **9** | Sets identified elements to `0` when match is found and returns appropriate `boolean` in both cases (*algorithm*) | Responses **can** still earn the point even if they<br>• identify elements incorrectly, as long as there is some conditional selection<br><br>Responses **will not** earn the point if they<br>• set only one identified element to `0`<br>• change more than 1 element besides the element at `row` and `col`<br>• return an incorrect value due to an early return<br>• return an incorrect value due to not returning after clearing once | **1 point** |
| | **Question-specific penalties** | | |
| | None | | |