

2025



AP[®] Computer Science A

Scoring Guidelines

AP[®] Computer Science A 2025 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 (`[]` vs. `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. | <pre>public int walkDogs(int hour) { int dogsToWalk = company.numAvailableDogs(hour); if (dogsToWalk > maxDogs) { dogsToWalk = maxDogs; } company.updateDogs(hour, dogsToWalk); return dogsToWalk; }</pre> | 4 points |
| b. | <pre>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; }</pre> | 5 points |

a. walkDogs

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

b. dogWalkShift

Scoring Criteria		Decision Rules	
5	Loops from <code>startHour</code> to <code>endHour</code> , inclusive	Responses can still earn the point even if they return from the loop early, as long as bounds would otherwise be correct	1 point
6	Calls <code>walkDogs</code> with <code>int</code> parameter (<i>in the context of a loop</i>)	Responses can still earn the point even if they - fail to save or use the returned value Responses will not earn the point if they - use an incorrect parameter type on any call to <code>walkDogs</code> - call the method on the class or on an object other than <code>this</code> (use of <code>this</code> is optional)	1 point
7	Calculates base pay for an hour	Responses can still earn the point even if they - fail to include the calculation in the loop - call <code>walkDogs</code> incorrectly	1 point
8	Calculates possible bonus for an hour's pay	Responses can still earn the point even if they - call <code>walkDogs</code> multiple times inside the loop - fail to include the calculation in the loop Responses will not earn the point if they - count the bonus twice when both conditions are true - count the bonus only when both conditions are true - count the bonus when it has not been earned - calculate bonus incorrectly	1 point
9	Accumulates total pay (<i>algorithm</i>)	Responses can still earn the point even if they - calculate base pay or bonus incorrectly - fail to return total pay (<i>return not assessed in this part</i>) Responses will not earn the point if they - fail to initialize variable used for total pay - call <code>walkDogs</code> multiple times inside the loop - do not accumulate base and bonus in the context of a loop - 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, “`ArrayList`” 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

```

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;
        }
    }
}

```

9 points

SignedText

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

6	Determines appropriate signature string in both cases (<i>algorithm</i>)	Responses can still earn the point even if they - fail to return a value in some cases (<i>return from <code>getSignature</code> not assessed</i>) Responses will not earn the point if they - construct the correct string but return something else - define another method inside the signature method or vice versa	1 point
7	Calls <code>String</code> methods using correct syntax throughout the class	Responses can still earn the point even if they - call <code>substring</code> only one time - call <code>indexOf</code>, <code>contains</code>, <code>charAt</code>, or other appropriate methods Responses will not earn the point if they - only call <code>length</code> and/or <code>equals</code> without using other <code>String</code> methods - fail to call at least one <code>String</code> method which accesses elements of a string	1 point
8	Identifies the three required cases for <code>addSignature</code> using appropriate conditions	Responses can still earn the point even if they return an incorrect string in one or more cases	1 point
9	<code>addSignature</code> returns appropriate <code>String</code> in all three cases (<i>algorithm</i>)	Responses can still earn the point even if they - identify one or more of the three cases incorrectly, as long as they are unambiguously no-match, match-start, and match-end - implement <code>getSignature</code> incorrectly Responses will not earn the point if they - call <code>getSignature</code> incorrectly, or incorrectly implement its functionality in <code>addSignature</code> - fail to identify three distinct cases - build correct strings but assign them to cases incorrectly - print the string instead of or in addition to returning it - define another method inside <code>addSignature</code> 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, “`ArrayList`” 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. `public Round(String[] names)` **4 points**

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

b. `public ArrayList<Match> buildMatches()` **5 points**

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

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

b. `buildMatches`

Scoring Criteria		Decision Rules	
5	Declares and initializes local <code>ArrayList</code> of <code>Match</code> objects	Responses will not earn the point if they fail to declare the <code>ArrayList</code>, even if they have other local variables declared	1 point
6	Initializes and maintains an index used to move from both ends of <code>competitorList</code> (<i>algorithm</i>)	Responses can still earn the point even if they - start at 0 instead of 1 or vice versa - make a bounds error with the "high" index, as long as it is counting down from the end of the list - 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 - fail to use the indices to construct a <code>Match</code>	1 point
7	Computes starting low index based on even/odd comparison	Responses can still earn the point even if they - compute the index incorrectly, as long as the even/odd cases start at different indices - call the <code>size</code> method incorrectly - modifies <code>competitorList</code> instead of skipping an index Responses will not earn the point if they - determine even and odd number of competitors incorrectly	1 point
8	Gets two elements of <code>competitorList</code> based on maintained index, constructs a <code>Match</code> object from them, and adds it to the local list	Responses can still earn the point even if they - omit <code>new</code> in the creation of the <code>Match</code> object (<i>use of <code>new</code> not assessed for this type</i>) - use incorrect indices Responses will not earn the point if they - access <code>competitorList</code> incorrectly - fail to create an object of type <code>Match</code> - 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 (<i>algorithm</i>)	Responses can still earn the point even if they - fail to return the <code>ArrayList</code> (<i>return not assessed in this question</i>) - determine even and odd number of competitors incorrectly, as long as the cases are unambiguous - fail to create or use a <code>Match</code> object Responses will not earn the point if they - fail to guard against even and odd numbers of competitors - include the first competitor for odd numbers of competitors or omit the first for even numbers - include too many matches (e.g. due to continuing past the middle of the list) - add indices instead of competitors - make syntactically incorrect call(s) to <code>size</code> or other <code>ArrayList</code> methods - permanently modify <code>competitorList</code>	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 (`x • ÷ ≤ ≥ <> ≠`)
- `[]` 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. | <pre>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; } } }</pre> | 4 points |
| b. | <pre>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; }</pre> | 5 points |

a. SumOrSameGame

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

b. clearPair

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