

2024



AP[®] Computer Science A

Scoring Guidelines

© 2024 College Board. College Board, Advanced Placement, AP, AP Central, and the acorn logo are registered trademarks of College Board. Visit College Board on the web: collegeboard.org.

AP Central is the official online home for the AP Program: apcentral.collegeboard.org.

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.*

Question 1: Methods and Control Structures**9 points****Canonical solution**

- (a)** `public void simulateOneDay(int numBirds)` **4 points**
- ```
{
 double condition = Math.random();
 if (condition < 0.05)
 {
 currentFood = 0;
 }
 else
 {
 int eachBirdEats = (int) (Math.random() * 41) + 10;
 int totalEaten = numBirds * eachBirdEats;
 if (totalEaten > currentFood)
 {
 currentFood = 0;
 }
 else
 {
 currentFood -= totalEaten;
 }
 }
}
```
- (b)** `public int simulateManyDays(int numBirds, int numDays)` **5 points**
- ```
{
    for (int daysSoFar = 0; daysSoFar < numDays; daysSoFar++)
    {
        if (currentFood == 0)
        {
            return daysSoFar;
        }
        simulateOneDay(numBirds);
    }
    return numDays;
}
```

(a) `simulateOneDay`

Scoring Criteria		Decision Rules	
1	Generates a random value	Responses can still earn the point even if they - fail to save or use the generated random value Responses will not earn the point if they - fail to call <code>Math.random</code>, or possibly the equivalent, at least one time - make any incorrect call to <code>Math.random</code>	1 point
2	Identifies two cases based on a comparison of a randomly generated value and some constant that implements a 5% probability	Responses can still earn the point even if they - reverse the 5/95 probability - compare <code>double</code> values using <code><=</code> or <code>>=</code> instead of <code><</code> or <code>></code> - incorrectly cast the 5/95 random value, as long as a suitable range is generated and the comparison divides that range appropriately - call <code>Math.random</code> incorrectly	1 point
3	Generates a random integer that is uniform in the range [10, 50]	Responses can still earn the point even if they call <code>Math.random</code> incorrectly	1 point
4	Scales for <code>numBirds</code> and subtracts appropriate amount from <code>currentFood</code> in all cases (<i>algorithm</i>)	Responses can still earn the point even if they - compare <code>double</code> values using <code><=</code> or <code>>=</code> instead of <code><</code> or <code>></code> Responses will not earn the point if they - reverse the 5/95 probability - incorrectly process the 5/95 range/comparison (e.g., by incorrect casting) - exit the method while <code>currentFood</code> has a negative value	1 point
Total for part (a)			4 points

(b) `simulateManyDays`

Scoring Criteria		Decision Rules	
5	Calls <code>simulateOneDay</code> with value of <code>numBirds</code>	Responses will not earn the point if they - fail to make at least one correct call to <code>simulateOneDay</code> - make any call to <code>simulateOneDay</code> on the class or on an object other than <code>this</code> (use of <code>this</code> is optional)	1 point
6	Loops over the simulation method call and guards that it runs at most <code>numDays</code> times	Responses can still earn the point even if they - call the method that simulates one day incorrectly - count the simulated days incorrectly, as long as the loop guard would work if the count were corrected - fail to guard against calls to the simulation method when <code>currentFood <= 0</code> Responses will not earn the point if they - fail to count the simulated days at all	1 point
7	Counts the number of times that the method that simulates one day is called with food available in the feeder (<i>algorithm</i>)	Responses can still earn the point even if they - simulate extra days when <code>currentFood</code> is 0, as long as the count is still correct Responses will not earn the point if they - fail to initialize the counter appropriately - compute the correct count but return something else - fail to test if food is available	1 point
8	Compares <code>currentFood</code> and 0	Responses can still earn the point even if they - fail to make the comparison in the context of a loop - use the result of the comparison incorrectly	1 point
9	Returns any <code>int</code> value, in all cases	Responses can still earn the point even if they return a constant	1 point
Total for part (b)			5 points
Total for question 1			9 points

Alternate canonical for part (b):

```
public int simulateManyDays(int numBirds, int numDays)
{
    int daysSoFar = 0;
    while (currentFood > 0 && daysSoFar < numDays)
    {
        simulateOneDay(numBirds);
        daysSoFar++;
    }

    return daysSoFar;
}
```

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.*

Question 2: Class**9 points****Canonical solution**

```
public class Scoreboard
{
    private String team1Name, team2Name;
    private int whoseTurn;
    private int score1, score2;

    public Scoreboard(String team1, String team2)
    {
        team1Name = team1;
        team2Name = team2;
        whoseTurn = 1;
        score1 = 0;
        score2 = 0;
    }

    public void recordPlay(int points)
    {
        if (points == 0)
        {
            if (whoseTurn == 1)
            {
                whoseTurn = 2;
            }
            else
            {
                whoseTurn = 1;
            }
        }
        else
        {
            if (whoseTurn == 1)
            {
                score1 += points;
            }
            else
            {
                score2 += points;
            }
        }
    }

    public String getScore()
    {
        String result = score1 + "-" + score2 + "-";
        if (whoseTurn == 1)
        {
            result += team1Name;
        }
        else
        {
            result += team2Name;
        }
        return result;
    }
}
```

9 points

Scoreboard

Scoring Criteria		Decision Rules	
1	Declares class header: <code>class Scoreboard</code>	Responses will not earn the point if they declare the class as something other than <code>public</code>	1 point
2	Declares at least one <code>private String</code> instance variable and one <code>private int</code> instance variable	Responses will not earn the point if they - declare any instance variable <code>static</code> - declare a variable outside the class	1 point
3	Declares constructor header: <code>Scoreboard(String ____, String ____)</code> and constructor initializes both team name instance variables using parameters	Responses can still earn the point even if they - declare instance variable(s) outside the class, or in the class within a method or constructor Responses will not earn the point if they - fail to declare or initialize instance variables for both team names - declare the constructor as something other than <code>public</code>	1 point
4	Declares method headers: <code>public void recordPlay(int ____) public String getScore()</code>	Responses will not earn the point if they - use incorrect method names - omit or declare incorrectly either method header - omit <code>public</code> in either method header or declare either method as something other than <code>public</code>	1 point
5	Recording method checks for parameter value of zero	Responses can still earn the point even if they use a method name inconsistent with the examples, as long as it is recognizably equivalent	1 point
6	Recording method increases at least one declared instance variable representing one team's score	Responses can still earn the point even if they - declare any instance variable incorrectly, outside the class, or in the class within a method or constructor - use something other than the parameter to update the instance variable - use a method name inconsistent with the examples, as long as it is recognizably equivalent	1 point

7	Recording method switches active team	Responses can still earn the point even if they - perform the switch in a method other than the recording method - store the switched active team in a local variable, as long as the switch occurs in both active team cases - use a method name inconsistent with the examples, as long as it is recognizably equivalent - perform the switch when the parameter is not zero	1 point
8	Recording method adds correct number of points to the active team's score (<i>algorithm</i>)	Responses can still earn the point even if they - fail to switch active team correctly - declare an instance variable that holds a team's score outside the class, or in the class within a method or constructor - use a method name inconsistent with the examples, as long as it is recognizably equivalent Responses will not earn the point if they - switch teams when the parameter is positive - fail to declare an instance variable to track the active team, initialize it incorrectly, or never change its value - add correct number of points for only one team - increase score by something other than the parameter - fail to declare instance variables to hold both teams' scores	1 point
9	Accessor method builds and returns specified string (<i>algorithm</i>)	Responses can still earn the point even if they - fail to declare instance variables and use variables from constructor or methods within the class - use a method name inconsistent with the examples, as long as it is recognizably equivalent Responses will not earn the point if they - omit the literal hyphens in the constructed string	1 point

Total for question 2 9 points

Alternate canonical:

```
public class Scoreboard
{
    private String team1Name, team2Name;
    private boolean isTeam1Active;
    private int score1, score2;

    public Scoreboard(String team1, String team2)
    {
        team1Name = team1;
        team2Name = team2;
        isTeam1Active = true;
        score1 = 0;
        score2 = 0;
    }

    public void recordPlay(int score)
    {
        if (score == 0)
        {
            isTeam1Active = !isTeam1Active;
        }
        else if (isTeam1Active)
        {
            score1 += score;
        }
        else
        {
            score2 += score;
        }
    }

    public String getScore()
    {
        String result = score1 + "-" + score2 + "-";
        if (isTeam1Active)
        {
            result += team1Name;
        }
        else
        {
            result += team2Name;
        }
        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.*

Question 3: Array / ArrayList**9 points****Canonical solution**

- (a) `public boolean isWordChain()` **3 points**
- ```
{
 for (int i = 1; i < wordList.size(); i++)
 {
 String current = wordList.get(i);
 String previous = wordList.get(i - 1);

 if (current.indexOf(previous) == -1)
 {
 return false;
 }
 }
 return true;
}
```
- (b) `public ArrayList<String> createList(String target)` **6 points**
- ```
{
    ArrayList<String> result = new ArrayList<String>();

    for (String current : wordList)
    {
        if (current.indexOf(target) == 0)
        {
            String newStr = current.substring(target.length());
            result.add(newStr);
        }
    }

    return result;
}
```

(a) `isWordChain`

Scoring Criteria		Decision Rules	
1	Accesses all adjacent pairs of <code>wordList</code> elements (<i>no bounds errors</i>)	Responses can still earn the point even if they - also access non-adjacent pairs of elements - return early, as long as bounds and indices would otherwise support accessing all necessary pairs Responses will not earn the point if they - fail to access elements of <code>wordList</code> correctly	1 point
2	Determines whether an element of the list contains a previous element of the list	Responses can still earn the point even if they - make just one comparison - fail to make the comparison in the context of a loop - compare every element to the first element of the list - access pairs of <code>wordList</code> elements incorrectly Responses will not earn the point if they - make an incorrect call to <code>indexOf</code> or use the <code>indexOf</code> return value incorrectly	1 point
3	Returns appropriate <code>boolean</code> in both cases (<i>algorithm</i>)	Responses can still earn the point even if they - incorrectly identify whether an element of <code>wordList</code> contains the previous element Responses will not earn the point if they - return an incorrect value due to an early return - fail to return <code>true</code> or <code>false</code>	1 point
Total for part (a)			3 points

(b) `createList`

Scoring Criteria		Decision Rules	
4	Declares and constructs an <code>ArrayList<String></code>	Responses will not earn the point if they fail to declare an <code>ArrayList</code>	1 point
5	Accesses all elements of <code>wordList</code> (no bounds errors)	Responses can still earn the point even if they - return early, as long as bounds and indices would otherwise support accessing all elements Responses will not earn the point if they - fail to access elements of <code>wordList</code> correctly	1 point
6	Identifies strings that begin with <code>target</code> (in the context of an <code>if</code>)	Responses can still earn the point even if they - access elements of <code>wordList</code> incorrectly Responses will not earn the point if they - call <code>String</code> methods incorrectly - identify all strings that contain <code>target</code> - use the <code>substring</code> method without a guard against an element too short to contain <code>target</code>	1 point
7	Constructs a <code>String</code> that is a copy of an element of the list with the correct number of initial characters removed	Responses can still earn the point even if they make a copy of a <code>wordList</code> element that does not start with <code>target</code> or is not long enough to contain <code>target</code>	1 point
8	Adds to the constructed list at least one <code>String</code> based on an element of the original list	Responses can still earn the point even if they - add an incorrectly constructed <code>String</code> - have not constructed a list Responses will not earn the point if they - call <code>add</code> incorrectly	1 point

9	Returns list containing all and only identified and revised strings in the appropriate order (<i>algorithm</i>)	Responses can still earn the point even if they - incorrectly identify strings beginning with <code>target</code> - call <code>add</code> incorrectly Responses will not earn the point if they - add the original, unrevised element to the list to be returned - modify <code>wordList</code> or any of its elements - return an incorrect value due to an early return	1 point
Total for part (b)			6 points
Total for question 3			9 points

Note that a correct part (b) solution could replace the `indexOf` call in the `if` statement with:

```
if (current.length() >= target.length() &&
    current.substring(0, target.length()).equals(target))
```

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.*

Question 4: 2D Arrays**9 points****Canonical solution**

- (a) `public Location getNextLoc(int row, int col)` **3 points**
- ```
{
 if (row == grid.length - 1)
 {
 return new Location(row, col + 1);
 }
 else if (col == grid[0].length - 1)
 {
 return new Location(row + 1, col);
 }
 else if (grid[row + 1][col] < grid[row][col + 1])
 {
 return new Location(row + 1, col);
 }
 else
 {
 return new Location(row, col + 1);
 }
}
```
- (b) `public int sumPath(int row, int col)` **6 points**
- ```
{
    int sum = 0;

    while (row < grid.length - 1 || col < grid[0].length - 1)
    {
        sum += grid[row][col];

        Location loc = getNextLoc(row, col);
        row = loc.getRow();
        col = loc.getCol();
    }
    return sum + grid[row][col];
}
```

(a) `getNextLoc`

Scoring Criteria		Decision Rules	
1	Guards against out-of-bounds access of <code>grid</code> elements	Responses can still earn the point even if they - fail to access any element of <code>grid</code> in this part, as long as the guard prevents the returned <code>Location</code> from being out of bounds Responses will not earn the point if they - return a <code>Location</code> that would be out of bounds	1 point
2	Accesses both an element of <code>grid</code> to the right and an element of <code>grid</code> below <code>row</code> and <code>col</code>	Responses can still earn the point even if they - access elements of <code>grid</code> out of bounds Responses will not earn the point if they - fail to access elements of <code>grid</code> correctly	1 point
3	Returns <code>Location</code> of appropriate grid element (<i>algorithm</i>)	Responses can still earn the point even if they - incorrectly guard against out-of-bounds access of grid elements Responses will not earn the point if they - call the <code>Location</code> constructor incorrectly - fail to consider all four cases	1 point
Total for part (a)			3 points

(b) `sumPath`

Scoring Criteria		Decision Rules	
4	Initializes and increases variable to store sum of grid values	Responses can still earn the point even if they - fail to initialize a local variable in a recursive solution, as long as an element of the grid is added to the recursive call Responses will not earn the point if they - initialize the variable to something other than <code>0</code> or an element of grid - increment the sum variable using something other than an element of grid	1 point
5	Determines the path based on successive calls to <code>getNextLoc</code> while current position is not the bottom-right position of grid (<i>no bounds errors</i>) (<i>algorithm</i>)	Responses can still earn the point even if they - fail to access an element of <code>grid</code> - call <code>getNextLoc</code> incorrectly - access row/column of next location incorrectly Responses will not earn the point if they - fail to call <code>getNextLoc</code> - fail to use row/column derived from <code>getNextLoc</code> return value in subsequent calls - stop loop early (omit required path locations) or late (violate <code>getNextLoc</code> precondition) due to incorrect boundary condition	1 point
6	Calls <code>getNextLoc</code> (<i>in the context of a loop</i>)	Responses can still earn the point even if they - call <code>getNextLoc</code> within an incorrect loop Responses will not earn the point if they - call <code>getNextLoc</code> on the class or on an object other than <code>this</code> (use of <code>this</code> is optional) - fail to call <code>getNextLoc</code> with two <code>int</code> arguments	1 point
7	Calls <code>getRow</code> and <code>getCol</code> on a <code>Location</code> object	Responses will not earn the point if they call either method incorrectly	1 point

8	Accesses a <code>grid</code> element at positions derived from the call to the next location method	Responses can still earn the point even if they - access an incorrect <code>grid</code> element - only access the grid at <code>row</code> and <code>col</code>, if the solution is recursive and the parameters of the recursive call are derived from a call to the next location method	1 point
9	Computes sum of values along path (<i>algorithm</i>)	Responses can still earn the point even if they - stop loop early or late due to incorrect boundary condition - fail to return the computed sum (<i>return not assessed in this part</i>) Responses will not earn the point if they - fail to include the first or last visited location in the sum	1 point
Total for part (b)			6 points
Total for question 4			9 points