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 1: Methods and Control Structures  9 points

Canonical solution

(a) public int scoreGuess(String guess)
    {
        int count = 0;
        for (int i = 0; i <= secret.length() - guess.length(); i++)
            if (secret.substring(i, i + guess.length()).equals(guess))
                count++;
        return count * guess.length() * guess.length();
    }

(b) public String findBetterGuess(String guess1, String guess2)
    {
        if (scoreGuess(guess1) > scoreGuess(guess2))
            return guess1;
        if (scoreGuess(guess2) > scoreGuess(guess1))
            return guess2;
        if (guess1.compareTo(guess2) > 0)
            return guess1;
        return guess2;
    }
<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Compares guess to a substring of secret</td>
<td>Responses can still earn the point even if they only call <code>secret.indexOf(guess)</code></td>
</tr>
<tr>
<td>2 Uses a substring of secret with correct length for comparison with guess</td>
<td>Responses can still earn the point even if they • only call <code>secret.indexOf(guess)</code> • use <code>==</code> instead of <code>equals</code></td>
</tr>
<tr>
<td>3 Loops through all necessary substrings of secret (no bounds errors)</td>
<td>Responses will not earn the point if they skip overlapping occurrences</td>
</tr>
<tr>
<td>4 Counts number of identified occurrences of guess within secret (in the context of a condition involving both secret and guess)</td>
<td>Responses can still earn the point even if they • initialize count incorrectly or not at all • identify occurrences incorrectly</td>
</tr>
<tr>
<td>5 Calculates and returns correct final score (algorithm)</td>
<td>Responses will not earn the point if they • initialize count incorrectly or not at all • fail to use a loop • fail to compare guess to multiple substrings of secret • count the same matching substring more than once • use a changed or incorrect guess length when computing the score</td>
</tr>
</tbody>
</table>

**Total for part (a)**  5 points
### (b) findBetterGuess

<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
</tr>
</thead>
</table>
| **6** Calls `scoreGuess` to get scores for `guess1` and `guess2` | Responses **will not** earn the point if they  
- fail to include parameters in the method calls  
- call the method on an object or class other than `this` |
| **7** Compares the scores | Responses **will not** earn the point if they  
- only compare using `==` or `!=`  
- fail to use the result of the comparison in a conditional statement |
| **8** Determines which of `guess1` and `guess2` is alphabetically greater | Responses **can still** earn the point even if they reverse the comparison  
- Responses **will not** earn the point if they  
  - reimplement `compareTo` incorrectly  
  - use result of `compareTo` as if `boolean` |
| **9** Returns the identified `guess1` or `guess2` *(algorithm)* | Responses **can still** earn the point even if they  
- call `scoreGuess` incorrectly  
- compare strings incorrectly  
- Responses **will not** earn the point if they  
  - reverse a comparison  
  - omit either comparison  
  - fail to return a guess in some case |

**Total for part (b) 4 points**

**Question-specific penalties**

None

**Total for question 1 9 points**
Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion ([] get)
w) Excessive 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 Design

Canonical solution

```java
public class CombinedTable {
    private SingleTable table1;
    private SingleTable table2;

    public CombinedTable(SingleTable tab1, SingleTable tab2) {
        table1 = tab1;
        table2 = tab2;
    }

    public boolean canSeat(int n) {
        if (table1.getNumSeats() + table2.getNumSeats() - 2 >= n) {
            return true;
        }
        else {
            return false;
        }
    }

    public double getDesirability() {
        if (table1.getHeight() == table2.getHeight()) {
            return (table1.getViewQuality() +
                    table2.getViewQuality()) / 2;
        } else {
            return ((table1.getViewQuality() +
                    table2.getViewQuality()) / 2) - 10;
        }
    }
}
```
<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
</tr>
</thead>
</table>
| **1** Declares class header:  
   class CombinedTable  
   and constructor header:  
   CombinedTable(SingleTable ___,  
   SingleTable ___)  
   *(must not be private)*  | Responses can still earn the point even if they declare the class header as class CombinedTable extends SingleTable |
| **2** Declares appropriate private instance variables including at least two SingleTable references  | Responses can still earn the point even if they declare an additional instance variable to cache the number of seats at the combined table  
   Responses will not earn the point if they  
   • declare and initialize local variables in the constructor instead of instance variables  
   • declare additional instance variable(s) that cache the desirability rating  
   • omit keyword `private`  
   • declare variables outside the class |
| **3** Constructor initializes instance variables using parameters  | Responses can still earn the point even if they declare and initialize local variables in the constructor instead of instance variables |
| **4** Declares header: public boolean canSeat(int ___)  |  |
| **5** Calls `getNumSeats` on a SingleTable object  | Responses can still earn the point even if they call `getNumSeats` on constructor parameters or local variables of type SingleTable in the constructor  
   Responses will not earn the point if they call the SingleTable accessor method on something other than a SingleTable object |
| **6** `canSeat(n)` returns true if and only if sum of seats of two tables - 2 >= n  | Responses can still earn the point even if they call `getNumSeats` incorrectly |
| **7** Declares header: public double getDesirability()  |  |
| **8** Calls `getHeight` and `getViewQuality` on SingleTable objects  | Responses can still earn the point even if they call `getHeight` or `getViewQuality` on constructor parameters or local variables of type SingleTable in the constructor |

© 2021 College Board
<table>
<thead>
<tr>
<th>Question</th>
<th>Description</th>
<th>Scoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td><code>getDesirability</code> computes average of constituent tables’ view desirabilities</td>
<td>Responses will not earn the point if they call the <code>SingleTable</code> accessor methods on something other than a <code>SingleTable</code> object. Responses can still earn the point even if they • call <code>getHeight</code> or <code>getViewQuality</code> on constructor parameters or local variables of type <code>SingleTable</code> in the constructor • fail to return the computed average <em>(return is not assessed)</em> Responses will not earn the point if they • fail to have an <code>if</code> statement and a correct calculation • choose the incorrect value (average vs. average – 10) based on evaluation of the <code>if</code> statement condition</td>
</tr>
</tbody>
</table>

### Question-specific penalties

None

### Total for question 2

9 points
Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion ([] get)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
x) Local variables used but none declared
y) Destruction of persistent data (e.g., changing value referenced by parameter)
z) Void method or constructor that returns a value

No Penalty

• Extraneous code with no side-effect (e.g., valid precondition check, no-op)
• Spelling/case discrepancies where there is no ambiguity*
• Local variable not declared provided other variables are declared in some part
• private or public qualifier on a local variable
• Missing public qualifier on class or constructor header
• Keyword used as an identifier
• Common mathematical symbols used for operators (× ÷ ≤ ≥ <> ≠)
• [] vs. () vs. <>
• = instead of == and vice versa
• length/size confusion for array, String, List, or ArrayList; with or without ( )
• Extraneous [] when referencing entire array
• [i,j] instead of [i][j]
• Extraneous size in array declaration, e.g., int[size] nums = new int[size];
• Missing ; where structure clearly conveys intent
• Missing {} where indentation clearly conveys intent
• Missing () on parameter-less method or constructor invocations
• Missing () around if or while conditions

*Spelling and case discrepancies for identifiers fall under the “No Penalty” category only if the correction can be unambiguously inferred from context, for example, “ArayList” instead of “ArrayList”. As a counterexample, note that if the code declares "int G=99, g=0;", then uses "while (G < 10) " instead of "while (g < 10)", the context does not allow for the reader to assume the use of the lower case variable.
Question 3: Array / ArrayList  9 points

Canonical solution

(a) public void addMembers(String[] names, int gradYear)
   {
      for (String n : names)
      {
         MemberInfo newM = new MemberInfo(n, gradYear, true);
         memberList.add(newM);
      }
   }

(b) public ArrayList<MemberInfo> removeMembers(int year)
   {
      ArrayList<MemberInfo> removed = new ArrayList<MemberInfo>();

      for (int i = memberList.size() - 1; i >= 0; i--)
      {
         if (memberList.get(i).getGradYear() <= year)
         {
            if (memberList.get(i).inGoodStanding())
            {
               removed.add(memberList.get(i));
            }
            memberList.remove(i);
         }
      }
      return removed;
   }
(a) \texttt{addMembers}

<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  \textit{Accesses all elements of names \textit{no bounds errors}}</td>
<td>Responses \textbf{will not} earn the point if they fail to access elements of the array, even if loop bounds are correct</td>
<td>1 point</td>
</tr>
<tr>
<td>2  \textit{Instantiates a MemberInfo object with name from array, provided year, and good standing}</td>
<td></td>
<td>1 point</td>
</tr>
<tr>
<td>3  \textit{Adds MemberInfo objects to memberList \textit{in the context of a loop}}</td>
<td>Responses \textbf{can} earn the point even if they instantiate MemberInfo objects incorrectly</td>
<td>1 point</td>
</tr>
</tbody>
</table>

\textbf{Total for part (a)} 3 points
### Scoring Criteria vs. Decision Rules

<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 <strong>Declares and initializes an ArrayList of MemberInfo objects</strong></td>
<td>Responses <strong>will not</strong> earn the point if they initialize the variable with a reference to the instance variable</td>
<td>1 point</td>
</tr>
<tr>
<td>5 <strong>Accesses all elements of memberList for potential removal</strong> (no bounds errors)</td>
<td>Responses <strong>will not</strong> earn the point if they • fail to use <code>get(i)</code> • fail to attempt to remove an element • skip an element • throw an exception due to removing</td>
<td>1 point</td>
</tr>
<tr>
<td>6 <strong>Calls getGradYear or inGoodStanding</strong></td>
<td>Responses <strong>can</strong> still earn the point even if they call only one of the methods</td>
<td>1 point</td>
</tr>
<tr>
<td>7 <strong>Distinguishes any three cases, based on graduation status and standing</strong></td>
<td>Responses <strong>will not</strong> earn the point if they fail to behave differently in all three cases</td>
<td>1 point</td>
</tr>
<tr>
<td>8 <strong>Identifies graduating members</strong></td>
<td>Responses <strong>can</strong> still earn the point even if they • fail to distinguish three cases • fail to access standing at all • access the graduating year incorrectly</td>
<td>1 point</td>
</tr>
<tr>
<td>9 <strong>Removes appropriate members from memberList and adds appropriate members to the ArrayList to be returned</strong></td>
<td>Responses <strong>can</strong> still earn the point even if they • call <code>getGradYear</code> or <code>inGoodStanding</code> incorrectly • access elements of memberList incorrectly • initialize the ArrayList incorrectly • fail to return the list that was built (return is not assessed)</td>
<td>1 point</td>
</tr>
</tbody>
</table>

**Total for part (b) 6 points**
<table>
<thead>
<tr>
<th>Question-specific penalties</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total for question 3</td>
<td>9 points</td>
</tr>
</tbody>
</table>
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 4: 2D Array

#### 9 points

**Canonical solution**

(a) 
```java
public static boolean isNonZeroRow(int[][] array2D, int r) {
    for (int col = 0; col < array2D[r].length; col++)
    {
        if (array2D[r][col] == 0)
        {
            return false;
        }
    }
    return true;
}
```

(b) 
```java
public static int[][] resize(int[][] array2D) {
    int numRows = array2D.length;
    int numCols = array2D[0].length;

    int[][] result = new int[numRows][numCols];
    int newRowIndex = 0;

    for (int r = 0; r < numRows; r++)
    {
        if (isNonZeroRow(array2D, r))
        {
            for (int c = 0; c < numCols; c++)
            {
                result[newRowIndex][c] = array2D[r][c];
            }
            newRowIndex++;
        }
    }
    return result;
}
```
(a) `isNonZero`

<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compares an item from <code>array2D</code> with 0</td>
<td>1 point</td>
</tr>
<tr>
<td></td>
<td>Responses will not earn the point if they fail to attempt the comparison, even if they access an item from <code>array2D</code></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Accesses every item from row <code>r</code> of 2D array <em>(no bounds errors)</em></td>
<td>1 point</td>
</tr>
<tr>
<td></td>
<td>Responses can still earn the point even if they return early from an otherwise correctly-bounded loop</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Returns <code>true</code> if and only if row contains no zeros</td>
<td>1 point</td>
</tr>
<tr>
<td></td>
<td>Responses can still earn the point even if they process a column of the 2D array rather than a row</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Responses will not earn the point if they fail to return a value in some cases</td>
<td></td>
</tr>
</tbody>
</table>

**Total for part (a)** 3 points
### Scoring Criteria

<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4</strong> Calls <em>numNonZeroRows</em> and <em>isNonZeroRow</em></td>
<td>Responses <strong>can</strong> still earn the point even if they fail to use or store the return value. Responses <strong>will not</strong> earn the point if they • include incorrect number or type of parameters • call methods on an object or class other than <em>ArrayResizer</em>.</td>
</tr>
<tr>
<td><strong>5</strong> Identifies rows with no zeros <em>(in the context of an if)</em></td>
<td>Responses <strong>can</strong> still earn the point even if they call <em>isNonZeroRow</em> incorrectly, if the row being tested is clearly identified (index or reference).</td>
</tr>
<tr>
<td><strong>6</strong> Declares and creates a new 2D array of the correct size</td>
<td>Response <strong>will not</strong> earn the point if they transpose the dimensions of the created array.</td>
</tr>
<tr>
<td><strong>7</strong> Maintains an index in the new array</td>
<td>Responses <strong>will not</strong> earn the point if they • fail to declare, initialize, and update a different index • maintain the index in a way that overwrites, skips, or duplicates rows.</td>
</tr>
<tr>
<td><strong>8</strong> Traverses all necessary elements of <em>array2D</em> <em>(no bounds errors)</em></td>
<td>Responses <strong>can</strong> still earn the point even if they • cause a bounds error by declaring and creating a new 2D array of an incorrect size • fail to maintain an index in the new array correctly, resulting in a bounds error • fail to access individual elements in a nested loop, if they access each row as an entire row. Responses <strong>will not</strong> earn the point if they transpose coordinates, leading to a bounds error and/or copying columns.</td>
</tr>
<tr>
<td><strong>9</strong> Copies all and only rows identified as having no zero elements into the new array</td>
<td>Responses <strong>can</strong> still earn the point even if they • copy a reference • identify rows incorrectly, if the logical sense can be determined and is correct • copy columns instead of rows, consistent with the dimensions of the created 2D array.</td>
</tr>
</tbody>
</table>
Responses will not earn the point if they
• remove or overwrite data from
array2D (instead of or in addition
to copying it to the new array)
• reverse the logical sense of which
rows to copy

<table>
<thead>
<tr>
<th>Question-specific penalties</th>
<th>Total for part (b)</th>
<th>6 points</th>
</tr>
</thead>
</table>
| -1 (u) Use array2D[].length to refer to
the number of columns in a row of the 2D
array | Total for question 4 | 9 points |

© 2021 College Board