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 • fail to update the instance variable `puzzle`, e.g., by redeclaring as a local variable • 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 • 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 `[1, 9]`	Responses **will not** earn the point if they • fail to call `Math.random` or an equivalent method • make any incorrect call to `Math.random` or an equivalent method • call `random` without `Math.` • 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 • 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 `puzzle` in rows `>=` `row` (*no bounds errors*)	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 `row` and `col` 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 `puzzle` 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 `row` and `col` 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 `0` *(in the context of a loop)*	Responses **can** still earn the point even if they • set only one identified element to `0` • incorrectly identify elements • 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 • 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 `0` • change more than 1 element besides the element at `row` and `col` • 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		

2019

AP® CollegeBoard

AP® Computer Science A

Scoring Guidelines

2019 SCORING GUIDELINES

Apply the question assessment rubric 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.*

2019 SCORING GUIDELINES

Question 1: Calendar

Part (a)	`numberOfLeapYears`	**5 points**

Intent: *Return the number of leap years in a range*

+1 Initializes a numeric variable

+1 Loops through each necessary year in the range

+1 Calls `isLeapYear` on some valid year in the range

+1 Updates count based on result of calling `isLeapYear`

+1 Returns count of leap years

Part (b)	`dayOfWeek`	**4 points**

Intent: *Return an integer representing the day of the week for a given date*

+1 Calls `firstDayOfYear`

+1 Calls `dayOfYear`

+1 Calculates the value representing the day of the week

+1 Returns the calculated value

Question-Specific Penalties

-1 (t) Static methods called with `this`.

2019 SCORING GUIDELINES

Question 1: Scoring Notes

Part (a) `numberOfLeapYears`			**5 points**
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Initializes a numeric variable		• use the variable for loop control only
+1	Loops through each necessary year in the range		• consider years outside the range
+1	Calls `isLeapYear` on some valid year in the range	• do not use a loop	
+1	Updates count based on result of calling `isLeapYear`	• do not use a loop • do not initialize the counter	• use result as a non-`boolean`
+1	Returns count of leap years	• loop from `year1` to `year2` incorrectly • do not initialize the counter	• do not use a loop • update or initialize the counter incorrectly • return early inside the loop
Part (b) `dayOfWeek`			**4 points**
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Calls `firstDayOfYear`		• do not use the given year
+1	Calls `dayOfYear`		• have arguments out of order
+1	Calculates the value representing the day of the week		• make any error in the calculation
+1	Returns the calculated value	• return the value from calling `firstDayOfYear` or `dayOfYear`	• return a constant value

2019 SCORING GUIDELINES

Question 1: Calendar

Part (a)

```
public static int numberOfLeapYears(int year1, int year2)
{
   int count = 0;
   for (int y = year1; y <= year2; y++)
   {
      if (isLeapYear(y))
      {
         count++;
      }
   }
   return count;
}
```

Part (b)

```
public static int dayOfWeek(int month, int day, int year)
{
   int startDay = firstDayOfYear(year);
   int nthDay = dayOfYear(month, day, year);
   int returnDay = (startDay + nthDay - 1) % 7;
   return returnDay;
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

2019 SCORING GUIDELINES

Question 2: Step Tracker

Class: `StepTracker`	**9 points**

Intent: *Define implementation of a class to record fitness data*

+1 Declares all appropriate `private` instance variables

+2 Constructor

- **+1** Declares header: `public StepTracker(int ___)`
- **+1** Uses parameter and appropriate values to initialize instance variables

+3 `addDailySteps` method

- **+1** Declares header: `public void addDailySteps(int ___)`
- **+1** Identifies active days and increments count
- **+1** Updates other instance variables appropriately

+1 `activeDays` method

- **+1** Declares and implements `public int activeDays()`

+2 `averageSteps` method

- **+1** Declares header: `public double averageSteps()`
- **+1** Returns calculated `double` average number of steps

2019 SCORING GUIDELINES

Question 2: Scoring Notes

Class `StepTracker`			**9 points**
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Declares all appropriate `private` instance variables		• omit keyword `private` • declare variables outside the class
+2	Constructor		
+1	Declares header: `public StepTracker(int ___)`	• omit keyword `public`	• declare method `private`
+1	Uses parameter and appropriate values to initialize instance variables	• initialize primitive instance variables to default values when declared	• fail to use the parameter to initialize some instance variable • fail to declare instance variables • initialize local variables instead of instance variables • assign variables to parameters
+3	`addDailySteps` method		
+1	Declares header: `public void addDailySteps(int ___)`	• omit keyword `public`	• declare method `private`
+1	Identifies active days and increments count	• put valid comparison erroneously in some other method	• fail to use the parameter as part of the comparison • fail to increment a count of active days • fail to increment an instance variable • compare parameter to some numeric constant
+1	Updates other instance variables appropriately		• update another instance variable only on active days • update another instance variable inappropriately • fail to update appropriate instance variable • update a local variable
+1	`activeDays` method		
+1	Declares and implements `public int activeDays()`	• return appropriate count of active days where the instance variables were updated improperly in `addDailySteps` or `activeDays`	• declare method `private` • return value that is not the number of active days • fail to return a value

2019 SCORING GUIDELINES

Question 2: Scoring Notes (continued)

Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+2	`averageSteps` method		
+1	Declares header: `public double averageSteps()`	• omit keyword `public`	• declare method `private`
+1	Returns calculated `double` average number of steps	• maintain instance variables improperly but calculate appropriate average • fail to handle the special case where no days are tracked	• use integer division • calculate something other than steps divided by days • fail to return

2019 SCORING GUIDELINES

Question 2: Step Tracker

```
public class StepTracker
{
   private int minSteps;
   private int totalSteps;
   private int numDays;
   private int numActiveDays;

   public StepTracker(int threshold)
   {
      minSteps = threshold;
      totalSteps = 0;
      numDays = 0;
      numActiveDays = 0;
   }

   public void addDailySteps(int steps)
   {
      totalSteps += steps;
      numDays++;
      if (steps >= minSteps)
      {
         numActiveDays++;
      }
   }

   public int activeDays()
   {
      return numActiveDays;
   }

   public double averageSteps()
   {
      if (numDays == 0)
      {
         return 0.0;
      }
      else
      {
         return (double) totalSteps / numDays;
      }
   }
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

2019 SCORING GUIDELINES

Question 3: Delimiters

Part (a)	`getDelimitersList`	**4 points**

Intent: *Store delimiters from an array in an* `ArrayList`

+1 Creates `ArrayList<String>`

+1 Accesses all elements in array `tokens` (*no bounds errors*)

+1 Compares strings in `tokens` with both instance variables (*must be in the context of a loop*)

+1 Adds delimiters into `ArrayList` in original order

Part (b)	`isBalanced`	**5 points**

Intent: *Determine whether open and close delimiters in an* `ArrayList` *are balanced*

+1 Initializes accumulator(s)

+1 Accesses all elements in `ArrayList delimiters` (*no bounds errors*)

+1 Compares strings in `delimiters` with instance variables and updates accumulator(s) accordingly

+1 Identifies and returns appropriate `boolean` value to implement one rule

+1 Identifies and returns appropriate `boolean` values for all cases

2019 SCORING GUIDELINES

Question 3: Scoring Notes

Part (a) `getDelimitersList`			**4 points**
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Creates `ArrayList<String>`	• omit `<String>`	• omit the keyword `new`
+1	Accesses all elements in array `tokens` (*no bounds errors*)	• return incorrectly inside the loop	• treat `tokens` as a single string • access elements of `tokens` as if from an `ArrayList` (e.g., `tokens.get(i)`)
+1	Compares strings in `tokens` with both instance variables (*must be in the context of a loop*)	• access elements of `tokens` as if from an `ArrayList` (e.g., `tokens.get(i)`)	• use `==` for string comparison • treat `tokens` as a single string
+1	Adds delimiters into `ArrayList` in original order	• add a delimiter by accessing `tokens` incorrectly (e.g., `tokens.get(i)`)	• add a token that is not a delimiter • do not maintain the original delimiter order
Part (b) `isBalanced`			**5 points**
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Initializes accumulator(s)	• initialize inside the loop	• initialize an accumulator variable but don't update it
+1	Accesses all elements in `ArrayList delimiters` (*no bounds errors*)	• return incorrectly inside the loop	• access elements of `delimiters` as if from an array (e.g., `delimiters[i]`)
+1	Compares strings in `delimiters` with instance variables and updates accumulator(s) accordingly	• access elements of `delimiters` as if from an array (e.g., `delimiters[i]`)	• use `==` for string comparison • adjust an accumulator without a guarding condition
+1	Identifies and returns appropriate `boolean` value to implement one rule	• check for more closing delimiters (inside a loop) and return `false` • return `true` if the number of open and close delimiters is the same, and `false` otherwise (after a loop)	
+1	Identifies and returns appropriate `boolean` values for all cases	• have correct logic with the exception of a loop bounds error, accessing elements as if from an array, or using `==` for string comparison	• initialize accumulator inside a loop • fail to check for more closing delimiters inside a loop

2019 SCORING GUIDELINES

Question 3: Delimiters

Part (a)

```
public ArrayList<String> getDelimitersList(String[] tokens)
{
   ArrayList<String> d = new ArrayList<String>();
   for (String str : tokens)
   {
      if (str.equals(openDel) || str.equals(closeDel))
      {
         d.add(str);
      }
   }
   return d;
}
```

Part (b)

```
public boolean isBalanced(ArrayList<String> delimiters)
{
   int openCount = 0;
   int closeCount = 0;

   for (String str : delimiters)
   {
      if (str.equals(openDel))
      {
         openCount++;
      }
      else
      {
         closeCount++;
      }

      if (closeCount > openCount)
      {
         return false;
      }
   }

   if (openCount == closeCount)
   {
      return true;
   }
   else
   {
      return false;
   }
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

2019 SCORING GUIDELINES

Question 4: Light Board

Part (a)	`LightBoard`	**4 points**

Intent: *Define implementation of a constructor that initializes a 2D array of lights*

+1 Creates a `new boolean[numRows][numCols]` and assigns to instance variable `lights`

+1 Accesses all elements in the created 2D array (*no bounds errors*)

+1 Computes the 40% probability

+1 Sets all values of 2D array based on computed probability

Part (b)	`evaluateLight`	**5 points**

Intent: *Evaluate the status of a light in a 2D array of lights*

+1 Accesses an element of `lights` as a `boolean` value in an expression

+1 Traverses specified `col` of a 2D array (*no bounds errors*)

+1 Counts the number of `true` values in the traversal

+1 Performs an even calculation and a multiple of three calculation

+1 Returns `true` or `false` according to all three rules

Question-Specific Penalties

-1 (z) Constructor returns a value

-1 (y) Destruction of persistent data

2019 SCORING GUIDELINES

Question 4: Scoring Notes

Part (a) `LightBoard`			**4 points**
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Creates a `new boolean[numRows][numCols]` and assigns to instance variable `lights`		• initialize a local variable that is never assigned to `lights` • omit the keyword `new` • use a type other than `boolean`
+1	Accesses all elements in the created 2D array (*no bounds errors*)	• fail to create `lights` but assume `lights[numRows][numCols]`	
+1	Computes the 40% probability	• use `Math.random() <= .4`	• incorrectly cast to `int`
+1	Sets all values of 2D array based on computed probability	• only assign `true` values	• compute a single probability but use it multiple times • reverse the sense of the comparison when assigning
Part (b) `evaluateLight`			**5 points**
Points	Rubric Criteria	Responses earn the point even if they...	Responses will not earn the point if they...
+1	Accesses an element of `lights` as a `boolean` value in an expression		• access `lights` as a type other than `boolean`
+1	Traverses specified `col` of a 2D array (*no bounds errors*)		
+1	Counts the number of `true` values in the traversal	• access too many or too few items in a single column • access a single row instead of a single column	• count an item more than once
+1	Performs an even calculation and a multiple of three calculation		• use `/` instead of `%`
+1	Returns `true` or `false` according to all three rules	• have an incorrect column count but use the correct logic	• fail to return a value in some case • implement counting loop more than once with one loop that is incorrect

2019 SCORING GUIDELINES

Question 4: Light Board

Part (a)

```
public LightBoard(int numRows, int numCols)
{
   lights = new boolean[numRows][numCols];

   for (int r = 0; r < numRows; r++)
   {
      for (int c = 0; c < numCols; c++)
      {
         double rnd = Math.random();
         lights[r][c] = rnd < 0.4;
      }
   }
}
```

Part (b)

```
public boolean evaluateLight(int row, int col)
{
   int numOn = 0;

   for (int r = 0; r < lights.length; r++)
   {
      if (lights[r][col])
      {
         numOn++;
      }
   }

   if (lights[row][col] && numOn % 2 == 0)
   {
      return false;
   }
   if (!lights[row][col] && numOn % 3 == 0)
   {
      return true;
   }
   return lights[row][col];
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.