4. This question involves a two-dimensional array of integers that represents a collection of randomly generated data. A partial declaration of the `Data` class is shown. You will write two methods of the `Data` class.

```
public class Data
{
   public static final int MAX = /* value not shown */;
   private int[][] grid;

   /** Fills all elements of grid with randomly generated values, as described in part (a)
    *  Precondition: grid is not null.
    *       grid has at least one element.
    */
   public void repopulate()
   {  /* to be implemented in part (a) */  }

   /** Returns the number of columns in grid that are in increasing order, as described
    *  in part (b)
    *  Precondition: grid is not null.
    *       grid has at least one element.
    */
   public int countIncreasingCols()
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write the `repopulate` method, which assigns a newly generated random value to each element of `grid`. Each value is computed to meet all of the following criteria, and all valid values must have an equal chance of being generated.

- The value is between 1 and `MAX`, inclusive.
- The value is divisible by 10.
- The value is not divisible by 100.

Complete the `repopulate` method.

```
/** Fills all elements of grid with randomly generated values, as described in part (a)
 *  Precondition: grid is not null.
 *       grid has at least one element.
 */
public void repopulate()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

(b) Write the `countIncreasingCols` method, which returns the number of columns in `grid` that are in increasing order. A column is considered to be in increasing order if the element in each row after the first row is greater than or equal to the element in the previous row. A column with only one row is considered to be in increasing order.

The following examples show the `countIncreasingCols` return values for possible contents of `grid`.

The return value for the following contents of `grid` is `1`, since the first column is in increasing order but the second and third columns are not.

10	50	40
20	40	20
30	50	30

The return value for the following contents of `grid` is `2`, since the first and third columns are in increasing order but the second and fourth columns are not.

10	540	440	440
220	450	440	190

Complete the `countIncreasingCols` method.

```
/** Returns the number of columns in grid that are in increasing order, as described
 *  in part (b)
 *  Precondition: grid is not null.
 *       grid has at least one element.
 */
public int countIncreasingCols()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Data

public static final int MAX = /* value not shown */
private int[][] grid

public void repopulate()
public int countIncreasingCols()
```

GO ON TO THE NEXT PAGE.

Question 4: 2D Array — 9 points

Canonical solution

(a) — **4 points**

```
public void repopulate()
{
   for (int row = 0; row < grid.length; row++)
   {
      for (int col = 0; col < grid[0].length; col++)
      {
         int rval = (int)(Math.random() * MAX) + 1;
         while (rval % 10 != 0 || rval % 100 == 0)
         {
            rval = (int)(Math.random() * MAX) + 1;
         }
         grid[row][col] = rval;
      }
   }
}
```

(b) — **5 points**

```
public int countIncreasingCols()
{
   int count = 0;

   for (int col = 0; col < grid[0].length; col++)
   {
      boolean ordered = true;

      for (int row = 1; row < grid.length; row++)
      {
         if (grid[row][col] < grid[row-1][col])
         {
            ordered = false;
         }
      }

      if (ordered)
      {
         count++;
      }
   }

   return count;
}
```

(a) `repopulate`

	Scoring Criteria	Decision Rules	
1	Traverses `grid` (*no bounds errors*)	Responses **will not** earn the point if they • fail to access an element of `grid` • access the elements of `grid` incorrectly • use enhanced `for` loops without using a `grid` element inside the loop	**1 point**
2	Generates a random integer in a range based on `MAX`	Responses **can** still earn the point even if they • assume or verify that `MAX >= 10` Responses **will not** earn the point if they • fail to cast to an `int`	**1 point**
3	Ensures that all produced values are divisible by 10 but not by 100	Responses **can** still earn the point even if they • fail to use a loop	**1 point**
4	Assigns appropriate values to all elements of `grid` (*algorithm*)	Responses **can** still earn the point even if they • assume or verify that `MAX >= 10` • produce some values that are not divisible by 10 or divisible by 100, if the range and distribution are otherwise correct Responses **will not** earn the point if they • use enhanced `for` loops and fail to maintain indices • produce values that are not equally distributed • produce values outside the specified range • exclude values that should be considered valid (other than errors in 10/100 handling)	**1 point**
		Total for part (a)	**4 points**

(b) `countIncreasingCols`

	Scoring Criteria	Decision Rules	
5	Traverses `grid` in column major order (*no loop header bounds errors*)	Responses **can** still earn the point even if they • access an out-of-bounds row or column index adjacent to the edge of the grid, if the loop bounds include only valid indices Responses **will not** earn the point if they • fail to access an element of `grid` • access the elements of `grid` incorrectly	**1 point**
6	Compares two elements in the same column of `grid`	Responses **can** still earn the point even if they • access elements of `grid` incorrectly • access elements in nonadjacent rows • compare elements with `==` • compare two elements in the same row instead of the same column	**1 point**
7	Determines whether a single column is in increasing order (*algorithm*)	Responses **can** still earn the point even if they • fail to reset variables in the outer loop before proceeding to the next column Responses **will not** earn the point if they • fail to access all pairs of adjacent elements in a single column • cause a bounds error by attempting to compare the first element of a column with a previous element or the last element of a column with a subsequent element • incorrectly identify a column with at least one pair of adjacent elements in decreasing order as increasing	**1 point**
8	Counts all columns that are identified as increasing (*algorithm*)	Responses **can** still earn the point even if they • detect increasing order for each row instead of each column • incorrectly identify increasing columns in the inner loop Responses **will not** earn the point if they • fail to initialize the counter	**1 point**

		• fail to reset variables in the outer loop causing subsequent runs of the inner loop to misidentify columns	
9	Returns calculated count of increasing columns	Responses **can** still earn the point even if they • calculate the count incorrectly	**1 point**
		Total for part (b)	**5 points**
	Question-specific penalties		
	None		
		Total for question 4	**9 points**

Alternate Canonical for Part (a)

```
public void repopulate()
{
   for (int row = 0; row < grid.length; row++)
   {
      for (int col = 0; col < grid[0].length; col++)
      {
         int rval = ((int)(Math.random() * (MAX / 10)) + 1) * 10;
         while (rval % 100 == 0)
         {
            rval = ((int)(Math.random() * (MAX / 10)) + 1) * 10;
         }
         grid[row][col] = rval;
      }
   }
}
```