Name _____ Period_____

1. The `LightBoard` class models a two-dimensional display of lights, where each light is either on or off, as represented by a Boolean value. You will implement a constructor to initialize the display and a method to evaluate a light.

```
public class LightBoard
{
     /** The lights on the board, where true represents on and false
      *  represents off.
      */

     private boolean[][] lights;

     /** Constructs a LightBoard object having numRows rows and numCols columns
      *  Precondition: numRows > 0, numCols > 0
      *  Postcondition: each light has a 60% probability of being set to on
      */

     public LightBoard(int numRows, int numCols)
     {     /* To be implemented in part (a)  */    }

     /** Evaluates a light in row index row and column index col
      *  and returns a status as described in part (b).
      *  Precondition: row and col are valid indexes in lights.
      */

     public boolean evaluateLight(int row, int col)
     { /* to be implemented in part (b) */ }

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

© Pluska

(a) Write the constructor for the `LightBoard` class, which initializes lights so that each light is set to on with a 60% probability. The notation `lights[r][c]` represents the array element at row `r` and column `c`.

Complete the `LightBoard` constructor below.

```
/** Constructs a LightBoard object having numRows rows and numCols columns.
 * Precondition: numRows > 0, numCols > 0
 * Postcondition: each light has a 60% probability of being set to on.
 */

public LightBoard(int numRows, int numCols) {


        lights = new boolean[numRows][numCols];
        for(int row = 0; row < lights.length; row++){
            for(int col = 0; col < lights[row].length; col++){
                lights[row][col] = (Math.random() < .6);
            }
        }









}
```

/4

Score _____/31

(b) Write the method `evaluateLight`, which computes and returns the status of a light at a given row and column based on the following rules.

1. If the light is off, return false if the number of lights in its column that are on is odd, including the current light.
2. If the light is on, return true if the number of lights in its column that are on is even, .
3. Otherwise, return the light's current status.

Class information for this question

```
public class LightBoard
private boolean[][] lights
public LightBoard(int numRows, int numCols)
public boolean evaluateLight(int row, int col)
```

Complete the evaluateLight method below.

```
 /** Evaluates a light in row index row and column index col and returns a status
  * as described in part (b).
  * Precondition: row and col are valid indexes in lights.
  */


public boolean evaluateLight(int row, int col) {


        int count = 0;
        for(int r = 0; r < lights.length; r++){
            if(lights[r][col]){
                count++;
            }
        }
        if(!lights[row][col] && count%2!=0){
            return false;
        }
        if(lights[row][col] && count%2 == 0){
            return true;
        }
        return lights[row][col];









}
```

/5

Score _____/31

2. This question involves manipulating a two-dimensional array of integers. You will write two static methods of the `ArrayResizer` class, which is shown below.

```
public class ArrayResizer {

    /**
     * Returns true if and only if every value in row r of array2D
     * is non-zero.
     * Precondition: r is a valid row index in array2D.
     * Postcondition: array2D is unchanged.
     */

    public static boolean isNonZeroRow(int[][] array2D, int r) {
        /* to be implemented in part (a) */ }

    /**
     * Returns the number of rows in array2D that contain all
     * non-zero values.
     * Postcondition: array2D is unchanged.
     */
    public static int numNonZeroRows(int[][] array2D) {
        /* implementation not shown */ }

    /**
     * Returns a new, possibly smaller, two-dimensional array that
     * contains only rows from array2D with no zeros, as described
     * in part (b).
     * Precondition: array2D contains at least one column
     * and at least one row with no zeros.
     * Postcondition: array2D is unchanged.
     */

    public static int[][] resize(int[][] array2D) {
        /* to be implemented in part (b) */ }
}
```

(a) Write the method `isNonZeroRow`, which returns true if and only if all elements in row r of a two-dimensional array array2D are not equal to zero.

For example, consider the following statement, which initializes a two-dimensional array.

```
int[][] arr = {{2, 1, 0},
               {1, 3, 2},
               {0, 0, 0},
               {4, 5, 6}};
```

Sample calls to isNonZeroRow are shown below.

Call to `isNonZeroRow`	Value Returned	Explanation
`ArrayResizer.isNonZeroRow(arr, 0)`	`false`	At least one value in row `0` is zero.
`ArrayResizer.isNonZeroRow(arr, 1)`	`true`	All values in row `1` are non-zero.
`ArrayResizer.isNonZeroRow(arr, 2)`	`false`	At least one value in row `2` is zero.
`ArrayResizer.isNonZeroRow(arr, 3)`	`true`	All values in row `3` are non-zero.

Complete the isNonZeroRow method.

```
/** Returns true if and only if every value in row r of array2D
* is non-zero.
* Precondition: r is a valid row index in array2D.
* Postcondition: array2D is unchanged.
*/
public static boolean isNonZeroRow(int[][] array2D, int r)
```

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

/3

Score _____/31

(b) Write the method `resize`, which returns a new two-dimensional array containing only rows from array2D with all non-zero values. The elements in the new array should appear in the same order as the order in which they appeared in the original array.

The following code segment initializes a two-dimensional array and calls the resize method.

```
int[][] arr = {{2, 1, 0},
               {1, 3, 2},
               {0, 0, 0},
               {4, 5, 6}};
int[][] smaller = ArrayResizer.resize(arr);
```

When the code segment completes, the following will be the contents of smaller.

```
{{1, 3, 2}, {4, 5, 6}}
```

A helper method, `numNonZeroRows`, has been provided for you. The method returns the number of rows in its two-dimensional array parameter that contain no zero values.

Complete the `resize` method. Assume that `isNonZeroRow` works as specified, regardless of what you wrote in part (a). You must use numNonZeroRows and isNonZeroRow appropriately to receive full credit.

```
/** Returns a new, possibly smaller, two-dimensional array that
* contains only rows from array2D with no zeros, as described
* in part (b).
* Precondition: array2D contains at least one column
* and at least one row with no zeros.
* Postcondition: array2D is unchanged.
*/
public static int[][] resize(int[][] array2D)
```

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

   int[][] result = new
      int[numNonZeroRows(array2D)][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;
}
```

/6

Score _____/31

3. This question involves reasoning about arrays of integers. You will write two static methods, both of which are in a class named `ArrayTester`.

```
public class ArrayTester
{

    /** Returns an array containing the elements of column c of arr2D in the same order as
     *  they appear in arr2D.
     *  Precondition: c is a valid column index in arr2D.
     *  Postcondition: arr2D is unchanged.
     */
    public static int[] getColumn(int[][] arr2D, int c)
    {  /* to be implemented in part (a) */  }


    /** Returns true if and only if every value in arr1 appears in arr2.
     *  Precondition: arr1 and arr2 have the same length.
     *  Postcondition: arr1 and arr2 are unchanged.
     */
    public static boolean hasAllValues(int[] arr1, int[] arr2)
    {
        /* To be implemented in part (b) */


    /** Returns true if arr contains any duplicate values;
     *          false otherwise.
     */
    public static boolean containsDuplicates(int[] arr)
    {  /* implementation not shown */  }


    /** Returns true if square is a Latin square as described in part (b);
     *          false otherwise.
     *  Precondition: square has an equal number of rows and columns.
     *                square has at least one row.
     */
    public static boolean isLatin(int[][] square)
    {
        /* To be implemented in part (c) */       }

}
```

(a) Write a static method `getColumn`, which returns a one-dimensional array containing the elements of a single column in a two-dimensional array. The elements in the returned array should be in the same order as they appear in the given column. The notation `arr2D[r][c]` represents the array element at row r and column c.

The following code segment initializes an array and calls the `getColumn` method.

```
int[][] arr2D = { { 0, 1, 2 },
                  { 3, 4, 5 },
                  { 6, 7, 8 },
                  { 9, 5, 3 } };

int[] result = ArrayTester.getColumn(arr2D, 1);
```

When the code segment has completed execution, the variable result will have the following contents.

```
result: {1, 4, 7, 5}
```

Complete method getColumn below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
```

```
public int[] getColumn(int arr2D[][], int col){

    int someCol[] = new int[arr2D.length];
    for (int r = 0; r < arr2D.length; r++)
    {
        someCol[r] = arr2D[r][col];
    }

    return someCol;
}
```

/4

Score _____/31

(b) Write the static method `hasAllValues` which returns true if all the elements in the first array are in the second array.

For the two arrays below, hasAllValues returns `true`

```
int arr1[] = {1, 2, 3, 4, 5}
int arr2[] = {5, 4, 3, 2, 1}
```

For the two arrays below, hasAllValues returns `false`

```
int arr1[] = {1, 2, 3, 4, 5}
int arr2[] = {5, 4, 3, 2, 0}
```

Complete the `hasAllValues` method below

```
/** Returns true if and only if every value in arr1 appears in arr2.
 *  Precondition: arr1 and arr2 have the same length.
 *  Postcondition: arr1 and arr2 are unchanged.
 */
public static boolean hasAllValues(int[] arr1, int[] arr2)
```

```
public boolean hasAllValues(int arr1[], int arr2[]){
    boolean found = false;
    for(int i = 0; i < arr1.length; i++){
        found = false;
        for(int j = 0; j < arr2.length; j++){
            if(arr1[i] == arr2[j]){
                found = true;
            }
        }
        if(found == false){
            return false;
        }
    }
    return found;
}
```

/4

Score _____/31

(c) Write the static method `isLatin`, which returns `true` if a given two-dimensional square array is a Latin square, and otherwise, returns `false`.

A two-dimensional square array of integers is a Latin square if the following conditions are true,

- The first row has no duplicate values.
- All values in the first row of the square appear in each row of the square.
- All values in the first row of the square appear in each column of the square.

Examples of Latin Squares

1	2	3
2	3	1
3	1	2

10	30	20	0
0	20	30	10
30	0	10	20
20	10	0	30

Examples that are NOT Latin Squares

1	2	1
2	1	1
1	1	2

Not a Latin square because the first row contains duplicate values

1	2	3
3	1	2
7	8	9

Not a Latin square because the elements of the first row do not all appear in the third row

1	2
1	2

Not a Latin square because the elements of the first row do not all appear in either column

The ArrayTester class provides two helper methods: `containsDuplicates` and `hasAllValues`. The method `containsDuplicates` returns `true` if the given one-dimensional array arr contains any duplicate values and `false` otherwise. The method `hasAllValues` returns `true` if and only if every value in arr1 appears in arr2. You can assume these methods work.

Complete method `isLatin` below. Assume that `getColumn and hasAllValues` work as specified, regardless of what you wrote above. You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *         false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

```
public boolean isLatin(int[][] arr2D){

        if (containsDuplicates(arr2D[0]))
        {
            return false;
        }
        for (int r = 1; r < arr2D.length; r++)
        {
            if (!hasAllValues(arr2D[0], arr2D[r]))
            {
                return false;
            }
        }
        for (int c = 0; c < arr2D[0].length; c++)
        {
            if (!hasAllValues(arr2D[0], getColumn(arr2D, c)))
            {
                return false;
            }
        }
    return true;
    }
```

/6

Score _____/31