|  |  |
| --- | --- |
| 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.  } | |
| (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 |
| (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 |

|  |  |  |
| --- | --- | --- |
| 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) \*/ }  } | | |
| 1. 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.    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) | | |
|  | | /3 |
| 1. 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) | | |
|  | /6 | |

|  |  |
| --- | --- |
| 3. This question involves reasoning about arrays of integers. You will write two static methods, both of which are in a class named ArrayTester.  The code consists of 28 lines. Line 1: public class Array Tester Line 2: left brace Line 3: forward slash, asterisk, asterisk, Returns an array containing the elements of column c of a r r 2 D in the same order as Line 4: asterisk, they appear in a r r 2 D Line 5: asterisk, Precondition, colon, c is a valid column index in a r r 2 D. Line 6: asterisk, Postcondition, colon, a r r 2 D is unchanged. Line 7: asterisk, forward slash Line 8: public static int, open bracket, close bracket, get Column, one word, lowercase g and capital C, henceforth referred to as get Column, open parenthesis, int, open bracket, close bracket, open bracket, close bracket, a r r 2 D, comma, int c, close parenthesis Line 9: left brace, forward slash, asterisk, to be implemented in part a, asterisk, forward slash, right brace Line 10: forward slash, asterisk, asterisk, Returns true if and only if every value in a r r 1 appears in a r r 2. Line 11: asterisk, Precondition, colon, a r r 1 and a r r 2 have the same length. Line 12: asterisk, Postcondition, colon, a r r 1 and a r r 2 are unchanged. Line 13: asterisk, forward slash Line 14: public static boolean has All Values, one word, lowercase h and capital A and capital V, henceforth referred to as has All Values, open parenthesis, int, open bracket, close bracket, a r r 1, comma, int, open bracket, close bracket, a r r 2, close parenthesis Line 15: left brace, forward slash, asterisk, implementation not shown, asterisk, forward slash, right brace Line 16: forward slash, asterisk, asterisk, Returns true if a r r contains any duplicate values, semicolon Line 17: asterisk, false otherwise. Line 18: asterisk, forward slash Line 19: public static boolean contains Duplicates, one word, lowercase c and capital D, henceforth referred to as contains Duplicates, open parenthesis, int, open bracket, close bracket, a r r, close parenthesis Line 20: left brace, forward slash, asterisk, implementation not shown, asterisk, forward slash, right brace Line 21: forward slash, asterisk, asterisk, Returns true if square is a Latin square as described in part b, semicolon Line 22: asterisk, false otherwise. Line 23: asterisk, Precondition, colon, square has an equal number of rows and columns. Line 24: asterisk, square has at least one row. Line 25: asterisk, forward slash Line 26: public static boolean is Latin, one word, lowercase i and capital L, henceforth referred to as is Latin, open parenthesis, int, open bracket, close bracket, open bracket, close bracket, square, close parenthesis Line 27: left brace, forward slash, asterisk, to be implemented in part b, asterisk, forward slash, right brace Line 28: right brace  /\* To be implemented in part (b) \*/  /\* To be implemented in part (c) \*/ | |
| 1. 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, open bracket, close bracket, open bracket, close bracket, a r r 2 D equals left brace, left brace, 0, comma 1, comma 2, right brace, comma left brace, 3, comma 4, comma 5, right brace, comma left brace, 6, comma 7 , comma 8, right brace, comma left brace, 9, comma 5, comma 3, right brace, right brace, semicolon int, open bracket, close bracket, result equals Array Tester dot get Column, open parenthesis, a r r 2 D comma 1, close parenthesis, semicolon  When the code segment has completed execution, the variable result will have the following contents.  result, colon, left brace, 1, comma 4, comma 7, comma 5, right brace  Complete method getColumn below.  The code consists of 6 lines. Line 1: forward slash, asterisk, asterisk, Returns an array containing the elements of column c of a r r 2 D in the same order as they Line 2: asterisk, appear in a r r 2 D. Line 3: asterisk, Precondition, colon, c is a valid column index in a r r 2 D. Line 4: asterisk, Postcondition, colon, a r r 2 D is unchanged. Line 5: asterisk, forward slash Line 6: public static int, open bracket, close bracket, get Column, open parenthesis, int, open bracket, close bracket, open bracket, close bracket, a r r 2 D, comma int c, close parenthesis      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 |
| 1. 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        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 |

|  |  |
| --- | --- |
| 1. 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.       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.    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 |