Unit 8: 2D Arrays

**Topic 2: 2D Array Algorithms**

| **Name:** |  | | |
| --- | --- | --- | --- |
| **Partner:** |  | | |

| Work with your partner to code up these methods; open and use the **U8T2 Lab Replit** for this lab (one partner starts, the other joins!) |
| --- |

| 2D Array Algorithms: Warm Ups | |
| --- | --- |
| Here is an incomplete Algorithm2DArrays **utility** class (all static methods); write the code for each static method and test each one using the test code provided.  public class Algorithm2DArrays  {  /\*\* Return true if target is found in the 2D array arr, false otherwise.  \* For example, if arr is {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}  \* and target is 5, this method returns true. If target is 15, returns false  \*  \* @param arr 2D array of ints  \* @param target target value to be searched for in arr  \* @return true if target is found at least once in arr; false otherwise  \*/  public static boolean isFound(int[][] arr, int target)  { **/\* implement this method! \*/** }  **TEST CODE for isFound:**  int[][] testArr1 = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};  System.out.println(Algorithm2DArrays.isFound(testArr1, 5));  System.out.println(Algorithm2DArrays.isFound(testArr1, 12));  System.out.println(Algorithm2DArrays.isFound(testArr1, 13));  **EXPECTED OUTPUT:**    Copy/paste your **isFound** method below:   |  | | --- |  [Sample implementation](#_m1p6nhy5a2rh) /\*\* Return the number of elements in arr that contain searchStr as a substring  \* For example, if arr is:  \* {{"hi", "bye", "howl", "goo"},  \* {"time", "owl", "you", "good"},  \* {"map", "low", "bow", "mom"}};  \* and searchStr is "ow", this method returns 4, because 4 strings in arr  \* contain "ow" as a substring.  \*  \* @param arr 2D array of Strings  \* @param searchStr value to be searched for in each String of arr  \* @return the number of elements that contain searchStr as a substring  \*/  public static int countElementsWithSubstring(String[][] arr, String searchStr)  { **/\* implement this method! \*/** }  **TEST CODE for countElementsWithSubstring:**  String[][] words = {{"hi", "bye", "howl", "goo"}, {"time", "owl", "you", "good"}, {"map", "low", "bow", "mom"}};  System.out.println(Algorithm2DArrays.countElementsWithSubstring(words, "o"));  System.out.println(Algorithm2DArrays.countElementsWithSubstring(words, "ow"));  System.out.println(Algorithm2DArrays.countElementsWithSubstring(words, "goo"));  System.out.println(Algorithm2DArrays.countElementsWithSubstring(words, "map"));  System.out.println(Algorithm2DArrays.countElementsWithSubstring(words, "gone"));  **EXPECTED OUTPUT:**    Copy/paste your **countElementsWithSubstring** method below:   |  | | --- |  [Sample implementation](#_vf3mbykp1ptq)   /\*\* Replaces all even integers in arr with 0; all odd integers are unchanged.  \* It then returns the number of changes that were made.  \*  \* Example: if arr is {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {4, 6, 8, 3, 5}}  \* then this method modifies arr to be:  \* {{1, 0, 3, 0, 5}, {0, 7, 0, 9, 0}, {0, 0, 0, 3, 5}}  \* and returns 8 (the number of even numbers replaced by 0)  \*  \* THIS METHOD MODIFIES THE ORIGINAL 2D ARRAY OBJECT!  \*  \* @param arr 2D array of ints  \* @return the number of changes made  \*/  public static int replaceEvensWithZero(int[][] arr)  { **/\* implement this method! \*/** }  **TEST CODE for replaceEvensWithZero:**  int[][] testArr2 = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {4, 6, 8, 3, 5}};    // method modifies original testArr2 object  int changes = Algorithm2DArrays.replaceEvensWithZero(testArr2);  System.out.println("Number of changes: " + changes);    System.out.println("2D Array modified:");  for (int[] row : testArr2) {  for (int val : row) {  System.out.print(val + " ");  }  System.out.println();  }  **EXPECTED OUTPUT:**    Copy/paste your **replaceEvensWithZero** method below:   |  | | --- |  [Sample implementation](#_4984m66iws06)   /\*\* Return the sum of all elements of arr that are in the one particular row,  \* specified by the row parameter  \*  \* PRECONDITION: 0 <= row < arr.length (i.e. row is guaranteed to be valid)  \*  \* @param arr 2D array of ints  \* @param row the row to add up  \* @return the sum of all elements in row    \*/  public static int sumForRow(int[][] arr, int row)  { **/\* implement this method! \*/** }  **TEST CODE for sumForRow:**  int[][] testArr3 = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {4, 6, 8, 3, 5}};  System.out.println(Algorithm2DArrays.sumForRow(testArr3, 0));  System.out.println(Algorithm2DArrays.sumForRow(testArr3, 1));  System.out.println(Algorithm2DArrays.sumForRow(testArr3, 2));  **EXPECTED OUTPUT:**    Copy/paste your **sumForRow** method below:   |  | | --- |  [Sample implementation](#_8lomyqb3g451)   /\*\* Return the sum of all elements of arr that are in the one particular column  \*  \* PRECONDITION: 0 <= col < arr[0].length (i.e. col is valid)  \*  \* @param arr 2D array of ints  \* @param col the column to add up  \* @return the sum of all elements in column  \*/  public static int sumForColumn(int[][] arr, int col)  { **/\* implement this method! \*/** }  **TEST CODE for sumForColumn:**  int[][] testArr4 = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {4, 6, 8, 3, 5}};  System.out.println(Algorithm2DArrays.sumForColumn(testArr4, 0));  System.out.println(Algorithm2DArrays.sumForColumn(testArr4, 1));  System.out.println(Algorithm2DArrays.sumForColumn(testArr4, 2));  System.out.println(Algorithm2DArrays.sumForColumn(testArr4, 3));  System.out.println(Algorithm2DArrays.sumForColumn(testArr4, 4));  **EXPECTED OUTPUT:**    Copy/paste your **sumForColumn** method below:   |  | | --- |  [Sample implementation](#_z24zks1ur9yg) /\*\* Creates a 2D array of ints of size n x m, where n is number of rows and  \* m is number of columns; the elements in the 2D array should be numbered 0  \* through n x m - 1, with the numbers filled top to bottom, left to right.  \* The method prints the 2D array "grid" of numbers, and also returns  \* the 2D array.  \*  \* For example, for rows = 3 and cols = 4, this method should print  \* (and return):  \* 0 1 2 3  \* 4 5 6 7  \* 8 9 10 11  \*  \* Print each element with a single space in between.  \*  \* @param rows number of rows  \* @param cols number of columns  \* @return the same 2D array "number grid" that gets printed by the method  \*/  public static int[][] printNumberGrid(int rows, int cols)  { **/\* implement this method! \*/** }  **TEST CODE for printNumberGrid:**  // the method will print the 2D array, and also return the 2D array  int[][] returnedArr = Algorithm2DArrays.printNumberGrid(5, 6);    System.out.println("-----------");    // print the returned array (number grid), just to test that the  // correct 2D array got returned -- this should print the same grid  // that is printed by the method  for (int[] row : returnedArr) {  for (int val : row) {  System.out.print(val + " ");  }  System.out.println();  }  **EXPECTED OUTPUT (note:** the test code above prints same number grid twice; the first time it’s being printed by the method itself, and the second time the test code is printing the returned 2D array)    Copy/paste your **printNumberGrid** method below:   |  | | --- |  [Sample implementation](#_vlhyfq5dfcmq)   /\*\* Creates and returns a square 2D array of Strings of size n x n; the  \* elements in the 2D array alternate between "X" and "O" (letter),  \* starting with "X"  \*  \* For example, if n = 5, this method should return the following 2D array  \* (with 5 rows and 5 columns):  \* X O X O X  \* O X O X O  \* X O X O X  \* O X O X O  \* X O X O X  \*  \* For example, if n = 6, this method should return the following 2D array  \* (with 6 rows and 6 columns):  \* X O X O X O  \* O X O X O X  \* X O X O X O  \* O X O X O X  \* X O X O X O  \* O X O X O X  \*  \* @param n number of rows and cols (creating square 2D array of size n x n)  \* @return a 2D array of Strings that alternate between "X" and "O"  \*/  public static String[][] gridOfXandO(int n)  { **/\* implement this method! \*/** }  **TEST CODE for gridOfXandO:**  String[][] xoGrid = Algorithm2DArrays.gridOfXandO(6);    for (String[] row : xoGrid) {  for (String val : row) {  System.out.print(val + " ");  }  System.out.println();  }  System.out.println("-----------------");  String[][] xoGrid2 = Algorithm2DArrays.gridOfXandO(7);    for (String[] row : xoGrid2) {  for (String val : row) {  System.out.print(val + " ");  }  System.out.println();  }  **EXPECTED OUTPUT:**    Copy/paste your **gridOfXandO** method below:   |  | | --- |  [two sample implementations](#_9n3nqhs4mt5) **(for this method, you will need to import ArrayList)**  /\*\* Searches through the 2D array wordChart and finds all strings with  \* length len; these strings are added to an ArrayList and returned.  \* If no strings have that length, return an empty ArrayList  \*  \* @param wordChart 2D array of Strings  \* @parram len the length of strings to search for  \* @return an ArrayList containing all strings in wordChart with length len  \*/  public static ArrayList<String> findStringsOfLength(String[][] wordChart, int len)  { **/\* implement this method! \*/** }  **TEST CODE for findStringsOfLength:**  String[][] words2 = {{"hi", "bye", "stuff", "goo"}, {"time", "up", "you", "good"}, {"map", "low", "bow", "mom"}};  ArrayList<String> returnedList = Algorithm2DArrays.findStringsOfLength(words2, 3);  System.out.println(returnedList);  ArrayList<String> returnedList2 = Algorithm2DArrays.findStringsOfLength(words2, 4);  System.out.println(returnedList2);  ArrayList<String> returnedList3 = Algorithm2DArrays.findStringsOfLength(words2, 6);  System.out.println(returnedList3);  **EXPECTED OUTPUT:**    Copy/paste your **findStringsOfLength** method below:   |  | | --- |  [Sample implementation](#_wr8pwddlmgdh) For the next method, copy/paste and add this Student class to your Replit:  public class Student {  private String name;  private int grade;    public Student(String name, int grade) {  this.name = name;  this.grade = grade;  }    public String getName() {  return name;  }    public int getGrade() {  return grade;  }  }  /\*\* calculates and returns the average class grade as a double for a  \* all Students in the 2D array seatingChart  \*  \* @param wordChart 2D array of Strings  \* @return an ArrayList containing all strings in wordChart with length len  \*/  public static double classAverage(Student[][] seatingChart)  { **/\* implement this method! \*/** }  **TEST CODE for classAverge:**  Student s1 = new Student("Abby", 95);  Student s2 = new Student("Barb", 98);  Student s3 = new Student("Cole", 91);  Student s4 = new Student("Dave", 92);  Student s5 = new Student("Ellie", 98);  Student s6 = new Student("Frank", 87);  Student s7 = new Student("George", 93);  Student s8 = new Student("Jim", 92);  Student s9 = new Student("Kara", 95);  Student s10 = new Student("Lenny", 98);  Student s11 = new Student("Mona", 100);  Student s12 = new Student("Nancy", 99);  Student[][] chart = {{s1, s2, s3}, {s4, s5, s6}, {s7, s8, s9}, {s10, s11, s12}};  double avg = Algorithm2DArrays.classAverage(chart);  System.out.println(avg);  **EXPECTED OUTPUT:**    Copy/paste your **classAverage** method below:   |  | | --- |  [Sample implementation](#_kqp10lmn24ux) | |

| 2D Array Algorithms: Challenges! | |
| --- | --- |
| /\*\* Returns a new 2D array that contains all values in the 2D array arr  \* in reverse, such that the last value in the last row becomes the first  \* value in the first row, and so on. This method does NOT modify arr, the  \* original 2D array object.  \*  \* Example: if arr is {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}  \* then this method returns the following 2D array:  \* {{12, 11, 10, 9}, {8, 7, 6, 5}, {4, 3, 2, 1}}  \*  \* @param arr 2D array of ints  \*/  public static int[][] reverse(int[][] arr)  { **/\* implement this method! \*/** }  **TEST CODE for reverse:**  int[][] testArr1 = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};  int[][] reversed = Algorithm2DArrays.reverse(testArr1);  System.out.println("Returned 2D array:");  for (int[] row : reversed) {  for (int val : row) {  System.out.print(val + " ");  }  System.out.println();  }  System.out.println("\nOriginal 2D array NOT modified:");  for (int[] row : testArr1) {  for (int val : row) {  System.out.print(val + " ");  }  System.out.println();  }  **EXPECTED OUTPUT:**    Copy/paste your **reverse** method below:   |  | | --- |   *Sample solution will be posted later*  /\*\* Returns true if any two consecutive elements, horizontally or vertically,  \* are equal, false otherwise. You do **not** need to check for "wrap around"  \* consecutiveness, such as if the last element in a row is the same as  \* the first:  \*  \* For example, if arr is:  \* 1, 2, **3**, 4  \* 2, 7, **3**, 5  \* 3, 4, 5, 6  \*  \* then this method returns **true** because there is a pair of consecutive  \* numbers (**vertically**)  \*  \* As another example, if arr is:  \* 1, 2, 3, 4  \* 2, 7, 8, 5  \* 3, **4**, **4**, 6  \*  \* then this method returns **true** because there is a pair of consecutive  \* numbers (**horizontally**)  \*  \* As another example, if arr is:  \* 1, 2, 3, 8  \* 2, 7, 8, 5  \* 3, 2, 5, 3  \*  \* then this method returns **false** because there are **not** two consecutive  \* numbers, vertically or horizontally, that are equal.  \*  \* NOTE: do not consider "wrap around" consecutiveness, in other words,  \* the 3's and 2's below are **not** considered consecutive:  \* 1, **2**, 3, 8  \* 2, 7, 8, 5  \* **3**, **2**, 5, **3**  \*  \* Similarly, equal numbers on a diagonal are **not** considered consecutive,  \* so the 8's below are not consecutive:  \* 1, 2, 3, **8**  \* 2, 7, **8**, 5  \* 3, 2, 5, 3  \*  \* @param arr 2D array of ints  \*/  public static boolean consecutive(int[][] arr)  { **/\* implement this method! \*/** }  **TEST CODE for consecutive:**  int[][] testArr1 = {{1, 2, 3, 4}, {2, 7, 3, 5}, {3, 4, 5, 6}};  int[][] testArr2 = {{1, 2, 3, 4}, {2, 7, 8, 5}, {3, 4, 4, 6}};  int[][] testArr3 = {{1, 2, 3, 8}, {2, 7, 8, 5}, {3, 2, 5, 3}};  int[][] testArr4 = {{1, 2, 9, 9}, {2, 7, 8, 5}, {3, 2, 5, 3}};  int[][] testArr5 = {{1, 2, 3, 8}, {0, 7, 8, 5}, {0, 2, 5, 3}};  System.out.println(Algorithm2DArrays.consecutive(testArr1));  System.out.println(Algorithm2DArrays.consecutive(testArr2));  System.out.println(Algorithm2DArrays.consecutive(testArr3));  System.out.println(Algorithm2DArrays.consecutive(testArr4));  System.out.println(Algorithm2DArrays.consecutive(testArr5));  **EXPECTED OUTPUT:**    Copy/paste your **consecutive** method below:   |  | | --- |   *Sample solution will be posted!*  /\*\* A magic square is a square grid filled with distinct positive integers  \* such that each element is a different integer, and the sum of the integers  \* in each row, column and diagonal is equal. This method should return true  \* if arr is a magic square.  \*  \* **PRECONDITION**: arr is a square 2D array with the same number of rows and  \* columns  \*  \* For example, if arr is:  \* 7, 2, 3  \* 0, 4, 8  \* 5, 6, 1  \*  \* then this method returns **true** because all numbers are unique (no duplicate  \* values) and each row, column, and diagonal add up to the same number (12)  \*  \* As another example, if arr is (note the 3 and 2 swapped):  \* 7, **3, 2**  \* 0, 4, 8  \* 5, 6, 1  \*  \* then this method returns **false** because the column sums no longer add up to  \* the same value  \*  \* As another example, if arr is:  \* 3, 3, 3  \* 3, 3, 3  \* 3, 3, 3  \* then this method returns **false** because although the sums are the same  \* along rows, columns, and diagonals, there are *duplicate* numbers.  \*  \* @param arr 2D array of ints  \*/  public static boolean magicSquare(int[][] arr)  { **/\* implement this method! \*/** }  **TEST CODE for magicSquare:**  int[][] testArr1 = {{7, 2, 3}, {0, 4, 8}, {5, 6, 1}};  int[][] testArr2 = {{7, 3, 2}, {0, 4, 8}, {5, 6, 1}};  int[][] testArr3 = {{3, 3, 3}, {3, 3, 3}, {3, 3, 3}};  int[][] testArr4 = {{4, 9, 2}, {3, 5, 7}, {8, 1, 6}};  int[][] testArr5 = {{2, 16, 13, 3}, {11, 5, 8, 10}, {7, 9, 12, 6}, {14, 4, 1, 15}};  int[][] testArr6 = {{2, 16, 13, 3}, {11, 5, 8, 10}, {7, 9, 12, 15}, {14, 4, 1, 6}};  int[][] testArr7 = {{2, 16, 13, 3}, {11, 5, 8, 10}, {7, 9, 12, 6}, {14, 4, 1, 8}};  System.out.println(Algorithm2DArrays.magicSquare(testArr1));  System.out.println(Algorithm2DArrays.magicSquare(testArr2));  System.out.println(Algorithm2DArrays.magicSquare(testArr3));  System.out.println(Algorithm2DArrays.magicSquare(testArr4));  System.out.println(Algorithm2DArrays.magicSquare(testArr5));  System.out.println(Algorithm2DArrays.magicSquare(testArr6));  System.out.println(Algorithm2DArrays.magicSquare(testArr7));  **EXPECTED OUTPUT:**    Copy/paste your **magicSquare** method below:   |  | | --- |   *Sample solution will be posted!* | |

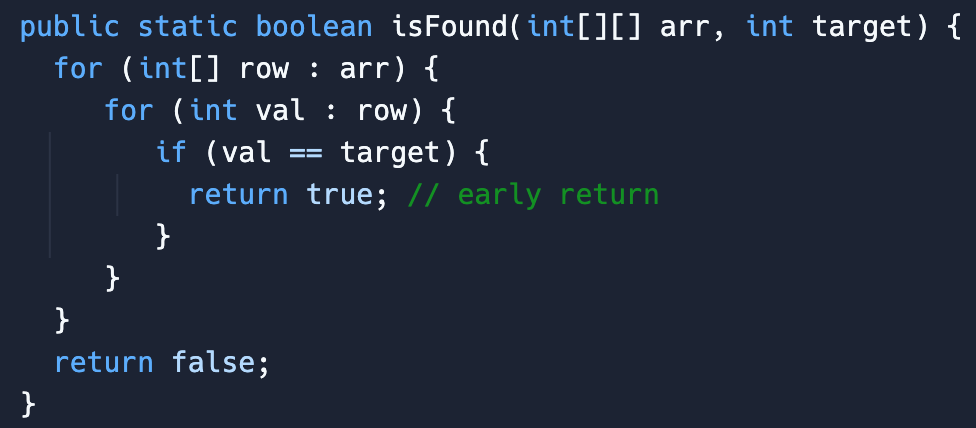
**Done!**

Submit in Google Classroom:

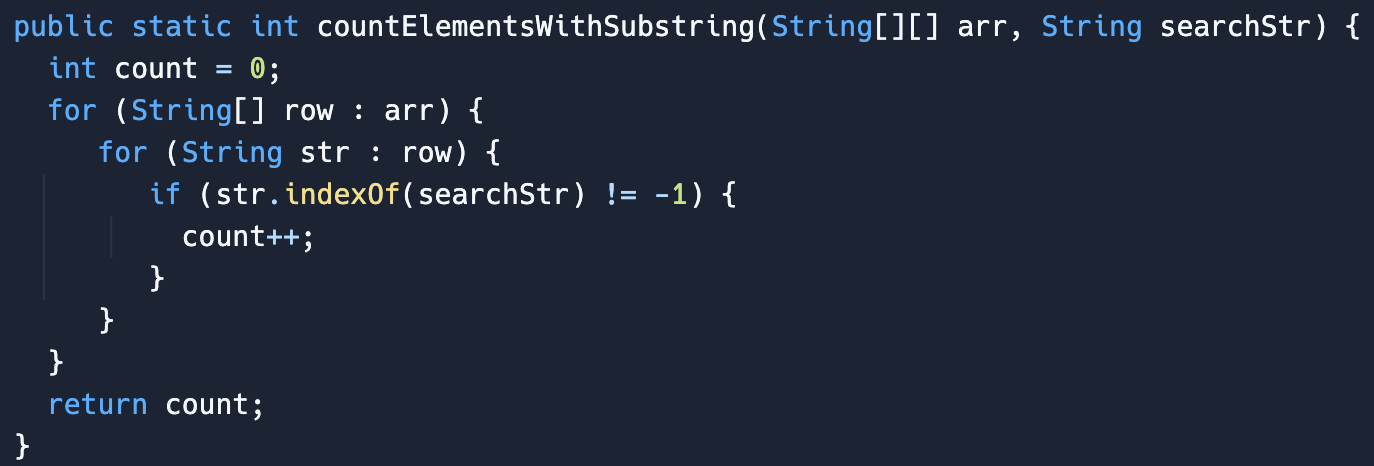


### Solution ([back](#_4wqnyfpr32du))

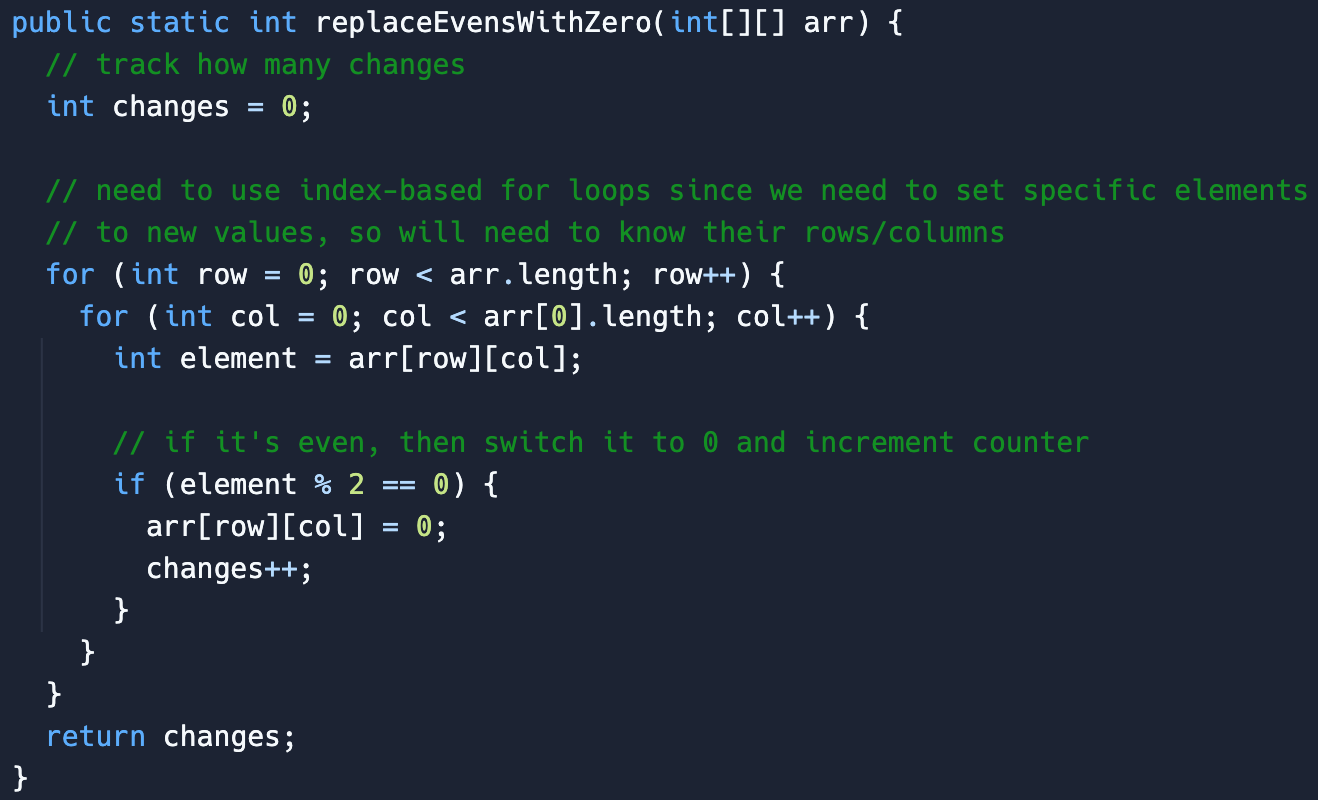
**Note the use of an enhanced for loop (this is preferred):**



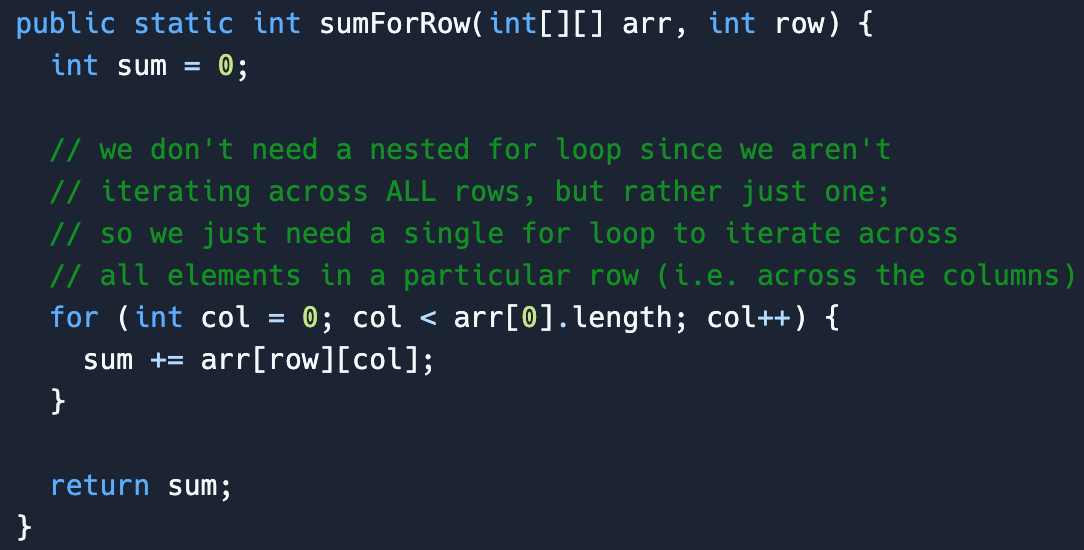
### Solution ([back](#_ibucqza0s00g))



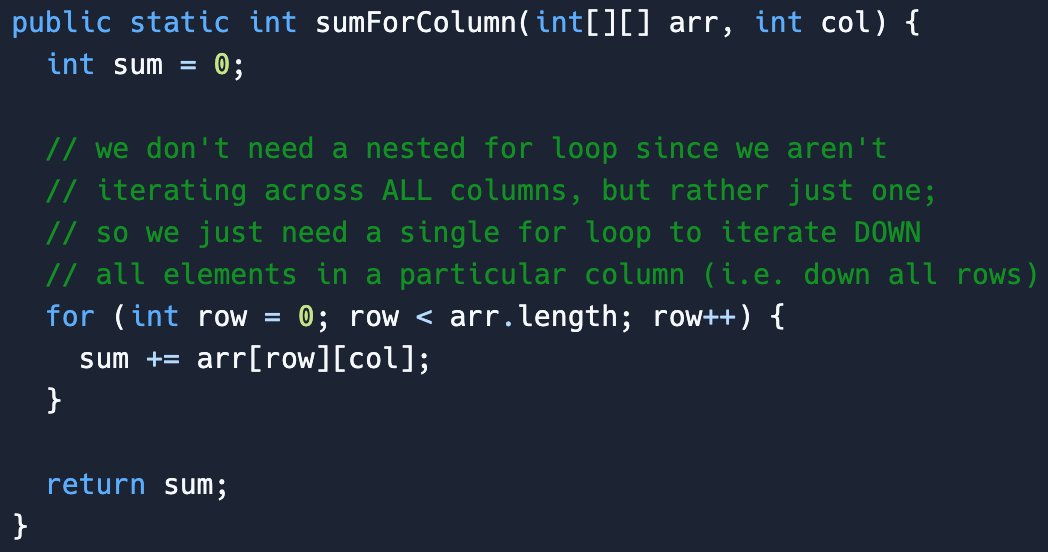
### Solution ([back](#_mwog8lx8iwjv))



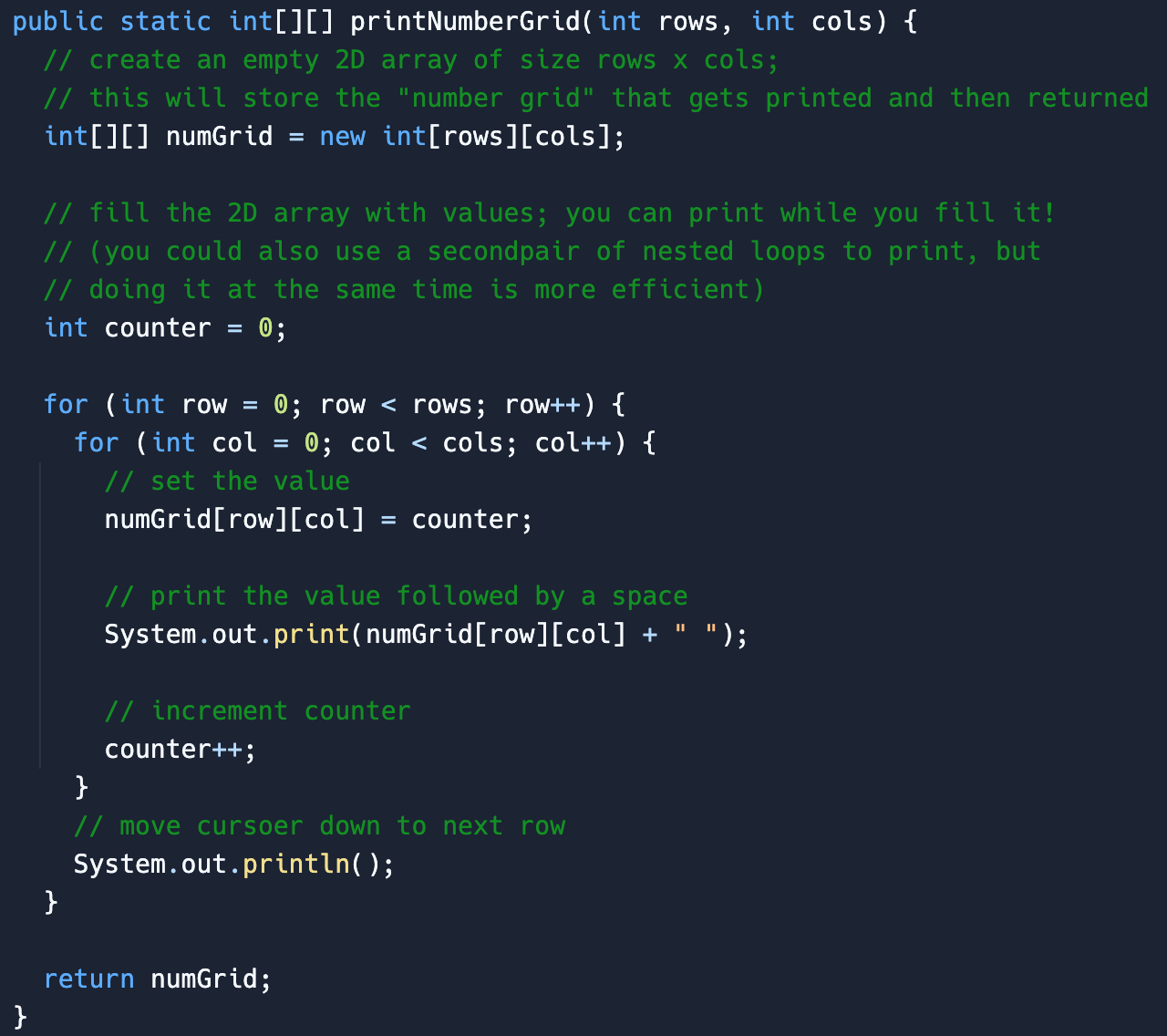
### Solution ([back](#_rxbtpr7d6740))



### Solution ([back](#_gbcse9xvbnac))



### Solution ([back](#_kfnoydo6f33))

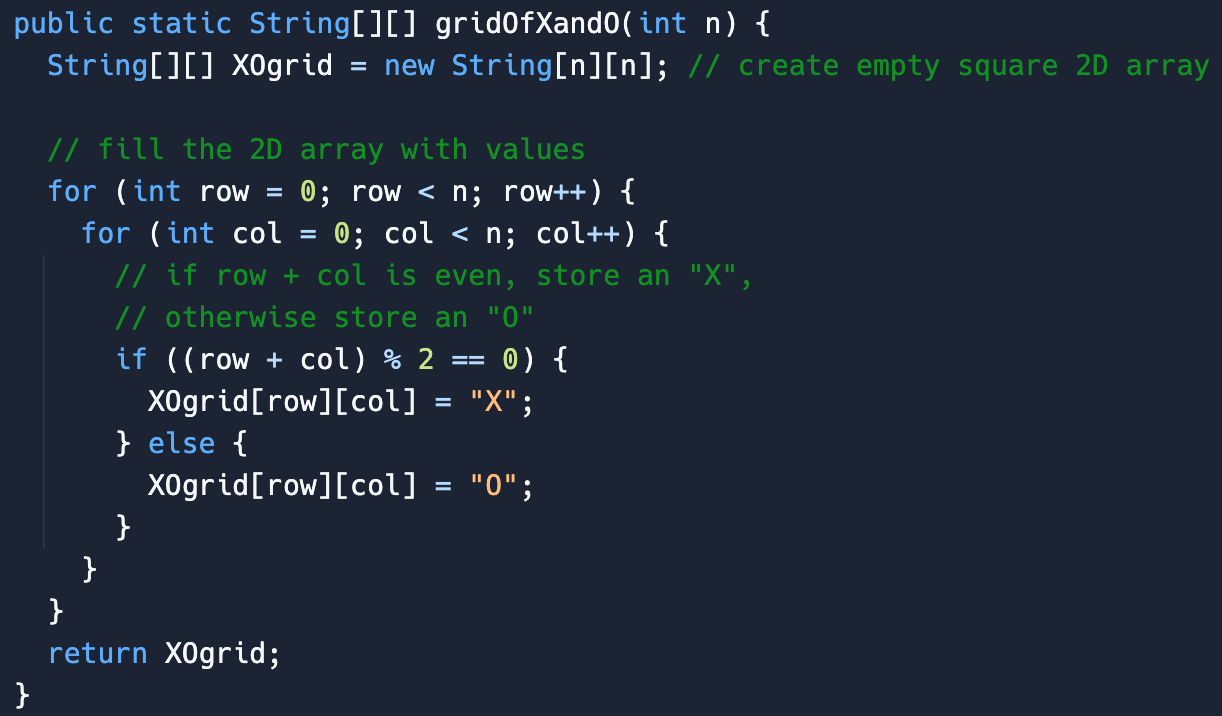


### Solutions ([back](#_kanmeccg629d))

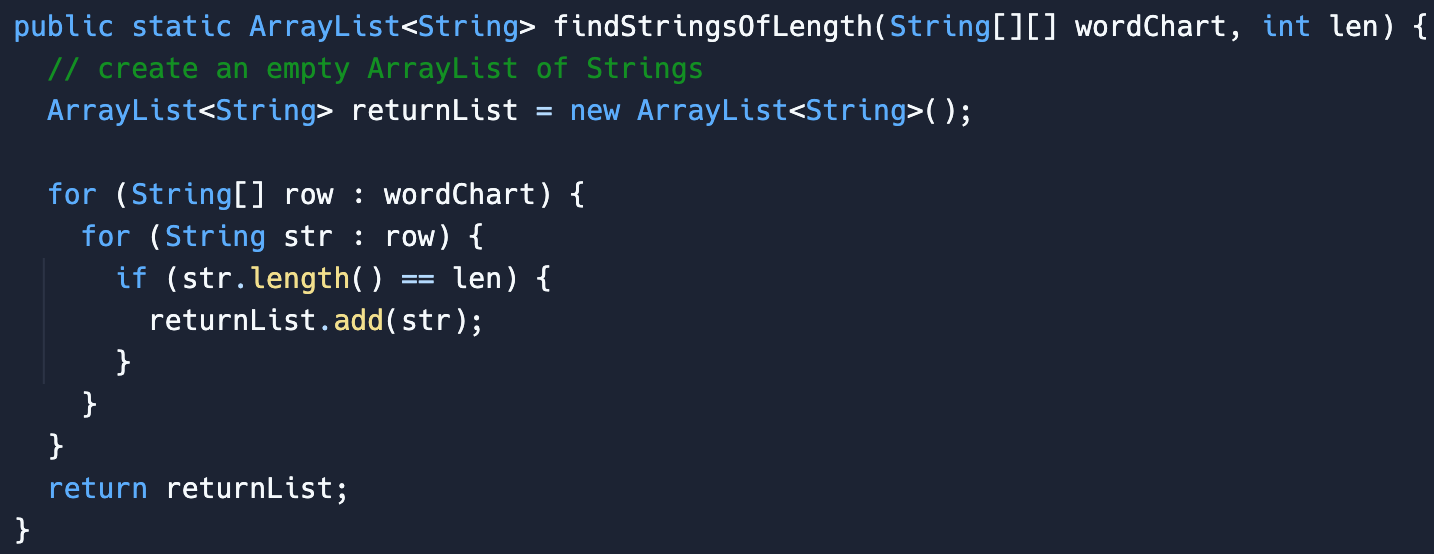
Here is one way to do it, where you swap back and forth between two different symbols (“X” and “O”)



Here is a bit more clever way to do it where you use the *sum* of row + col, and if the sum is even, print “X”, if it’s odd, print “O” (pretty slick!):



### Solution ([back](#_hz5l8dn5w6lc))



### Solution ([back](#_gmnflcbp6a9u))

