Unit 8: 2D Arrays

**Topic 2: Traversing 2D Arrays**

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

| Traversing 2D arrays | | |
| --- | --- | --- |
| **1.** Open your **U8T1 lab** from yesterday (in IntelliJ or Replit); you are going to add to it.  **2.** Clear out all code you had in the main method of your runner class.  **3.** In the main method, **copy/paste** the following code, then **complete** the WRITE CODE HERE sections as specified.  **/\* --- 1 --- \*/**  System.out.println("PROBLEM 1:");  int[][] **nums** = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}};  // write code below that uses nested INDEX-BASED for loops  // to print each element in **nums** in ROW-MAJOR order;  // print each number on its own line  // write code here  System.out.println("---------");  // now, write code below that uses nested ENHANCED for loops  // to print each element in **nums** in ROW-MAJOR order again;  // print all the numbers in a row on the same line  // with a space in between  // write code here  System.out.println("---------");  // write code below that uses nested for loops  // to print each element in **nums** in COLUMN-MAJOR order;  // print all the numbers in a COLUMN on the same line  // with a space in between  // write code here  **/\* --- 2 --- \*/**  System.out.println("---------");  System.out.println("PROBLEM 2:");  String[][] **animals** = {  {"anteater", "bird"},  {"camel", "dog"},  {"elephant", "giraffe"},  {"hyena", "jackal"}  };  // write code using a nested loop to update each element  // in **animals** to its plural form, e.g. bird --> birds;  // do not print anything here (just update)  // write code here  // write code to print out the 2D array in  // row-major order, showing each animal in **animals**  // inside a bracketed "enclosure" using  // "|" (sort of like fences separation animal  // pens in a zoo)  // write code here | | **Expected Output:** |
| **Copy/paste your code below:**   |  | | --- |  [Sample solutions](#_bp2x3iicdvdl)  | **Explain:** Is it possible to use a nested *enhanced* for loops to easily print a 2D array in *column-major* order?  As an example, could you use a nested *enhanced* for loop to print the nums array in problem 1 above in column-major order like this:    Show how or explain why not! |  | | --- | --- | | | |

**Lab continues on the next page**

| 2D Array Methods! | |
| --- | --- |
| 1. Open your FunWith2DArrays class from the last lab. 2. Add a static method named asciiPrinter that takes a 2D array of Strings, where each String is a *single-character*, and prints each element in each row directly adjacent to each other, with no spaces in between. For example, if the parameter to this method is the following 2D array named letters (note that some of the single-character string elements are spaces):     …then a call to FunWith2DArrays .asciiPrinter(letters) should print the following:    **TEST CODE:**   | // TEST CODE FOR asciiPrinter  String[][] ascii = {  {" ", " ", "\_", "\_", "\_", " ", " "},  {" ", "(", "o", " ", "o", ")", " "},  {"(", " ", " ", "V", " ", " ", ")"},  {" ", "-", "m", "-", "m", "-", " "},  };  FunWith2DArrays.asciiPrinter(ascii); | | --- |   **EXPECTED OUTPUT:**     1. Add a static method named average that takes a 2D array of ints as a parameter and returns the average (as a double) of all elements in the 2D array.   **TEST CODE:**   | // TEST CODE FOR average  int[][] testArr1 = {{1, 2, 3}, {4, 5, 6}};  System.out.println(FunWith2DArrays.average(testArr1));  int[][] testArr2 = {{1, 2, 3, 4}, {7, 8, 9, 10}};  System.out.println(FunWith2DArrays.average(testArr2));  int[][] testArr3 = {{4, 5, 2}, {1, 9, 7}, {8, 10, 13}};  System.out.println(FunWith2DArrays.average(testArr3));  int[][] testArr4 = {{4}, {7}, {8}, {2}};  System.out.println(FunWith2DArrays.average(testArr4));  int[][] testArr5 = {{4}};  System.out.println(FunWith2DArrays.average(testArr5)); | | --- |   **EXPECTED OUTPUT:**     1. Add a static method named indexFound that takes a 2D array of Strings and a target String as parameters and returns a 1D array of ints containing exactly two elements: the first element should indicate the *row* and the second element should indicate the *column* where the target is found in the 2D array. For example, if the 2D array is {{"a", "b", "c"}, {"d", "e", "f"}} and the target is "f", then this method should return [1, 2], since "f" is found in row 1, column 2 (0-indexed). If the target is *not* found, the method should return [-1, -1]   **PRECONDITION**: the target is only in the 2D array at most once  **TEST CODE:**   | // TEST CODE FOR indexFound  System.out.println("--------------");  String[][] letters = {{"a", "b", "c"}, {"d", "e", "f"}};  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(letters, "f")));  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(letters, "d")));  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(letters, "b")));  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(letters, "g")));  System.out.println("--------------");  String[][] words = {{"hi", "bye", "stuff", "go"}, {"time", "up", "you", "good"}, {"map", "low", "bow", "mom"}};  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(words, "stuff")));  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(words, "good")));  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(words, "map")));  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(words, "cat")));  System.out.println("--------------");  String[][] words2 = {{"time", "up", "bye"}, {"hi", "hit", "up"}, {"map", "bam", "low"}, {"bow", "mom", "joy"}};  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(words2, "time")));  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(words2, "joy")));  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(words2, "bam")));  System.out.println(Arrays.toString(FunWith2DArrays.indexFound(words2, "cat"))); | | --- |   **EXPECTED OUTPUT:**     1. Add a static method named split that takes a 2D array of ints, a row number, and a column number (both ints), and returns a *portion* of a 2D array based on the specified row and column.   For example, if nums is {{1,2,3,4},{5,6,7,8},{9,10,11,12}},  the call FunWith2DArrays.split(nums, 1, 2) would return all elements up *through* row 1, column 2, as a new 2D array:  **[1, 2, 3]**  **[5, 6, 7]**  If either the row or column value that is passed exceeds the boundaries of the 2D array, then return a 2D array with *no* elements, in other words, a 2D array with 0 rows and 0 columns.  **TEST CODE:**   | // TEST CODE FOR split  System.out.println("-------test1--------");  int[][] testArr6 = {{1, 2, 3}, {4, 5, 6}};  int[][] t1 = FunWith2DArrays.split(testArr6, 1, 1);  for (int[] row : t1) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("-------test2--------");  int[][] testArr7 = {{10, 11, 12, 13}, {20, 21, 22, 23}, {30, 31, 32, 33}};  int[][] t2 = FunWith2DArrays.split(testArr7, 2, 1);  for (int[] row : t2) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("-------test3--------");  int[][] testArr8 = {  {1, 2, 3, 4, 5, 6},  {7, 8, 9, 10, 11, 12},  {13, 14, 15, 16, 17, 18},  {19, 20, 21, 22, 23, 24}  };  int[][] t3 = FunWith2DArrays.split(testArr8, 3, 2);  for (int[] row : t3) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("-------test4--------");  int[][] t4 = FunWith2DArrays.split(testArr8, 2, 5);  for (int[] row : t4) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("-------test5--------");  int[][] t8 = FunWith2DArrays.split(testArr8, 0, 4);  for (int[] row : t8) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("-------test6--------");  int[][] t9 = FunWith2DArrays.split(testArr8, 0, 0);  for (int[] row : t9) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("-------test7--------");  int[][] t5 = FunWith2DArrays.split(testArr8, 3, 5);  for (int[] row : t5) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("-------test8--------");  int[][] t10 = FunWith2DArrays.split(testArr8, 3, 0);  for (int[] row : t10) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("-------test9--------");  int[][] t6 = FunWith2DArrays.split(testArr8, 4, 2);  for (int[] row : t6) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("-------test10-------");  int[][] t7 = FunWith2DArrays.split(testArr8, 2, 6);  for (int[] row : t7) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  } | | --- |   **EXPECTED OUTPUT:**    **NOTE: test9** and **test10** show that nothing is printed; this is correct, in that the returned 2D arrays from the method should have 0 rows and 0 columns, in other words no elements at all!   1. Add a static method named invertSquare that takes a 2D array of ints and returns an "inverted" version of that 2D array. "Inverting a 2D array" means that each row of the 2D array becomes a column, and each column becomes a row. The first row becomes the first column, and so on. This method returns a new 2D array and should not modify the original 2D array being passed in     For example, if you were to …the returned "inverted" array  invert this 2D array… would look like this:  [1, 2, 3, 4] [1, 5, 9]  [5, 6, 7, 8] [2, 6, 10]  [9, 10, 11, 12] [3, 7, 11]  [4, 8, 12]  **POSTCONDITION**: the array passed in as a parameter should **not** be modified.  **TEST CODE:**   | // TEST CODE FOR invert  System.out.println("---invert test1---");  int[][] test1 = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};  int[][] ts1 = FunWith2DArrays.invert(test1);  for (int[] row : ts1) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("orig array not modified:");  for (int[] row : test1) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("---invert test2---");  int[][] test2 = {  {1, 2, 3, 4, 5, 6},  {7, 8, 9, 10, 11, 12},  {13, 14, 15, 16, 17, 18},  {19, 20, 21, 22, 23, 24}  };  int[][] ts2 = FunWith2DArrays.invert(test2);  for (int[] row : ts2) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  System.out.println("---invert test3---");  int[][] test3 = {{10}};  int[][] ts3 = FunWith2DArrays.invert(test3);  for (int[] row : ts3) {  for (int num : row) {  System.out.print(num + " ");  }  System.out.println();  }  } | | --- |   **EXPECTED OUTPUT:**    Copy/paste the code for your **five new** methods below: | |
|  | |

###### [Sample solutions](#_yxqadw91h1dd)

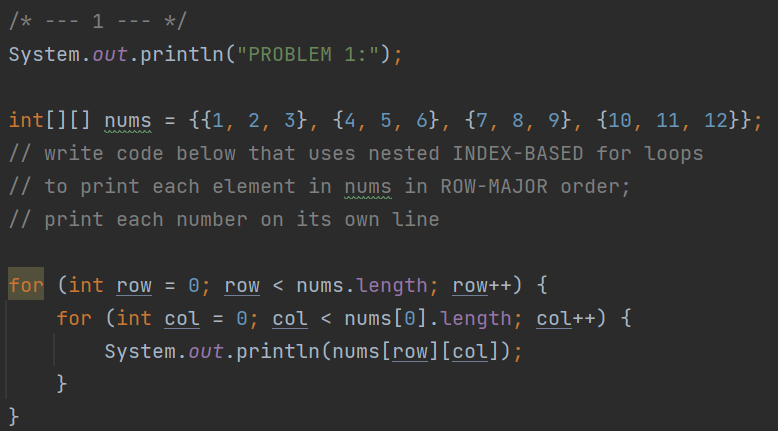
| Were you able to figure **all** of them out without peeking at the solutions? If so, awesome! 😎 If not, which methods did you have to peek at? |  |
| --- | --- |

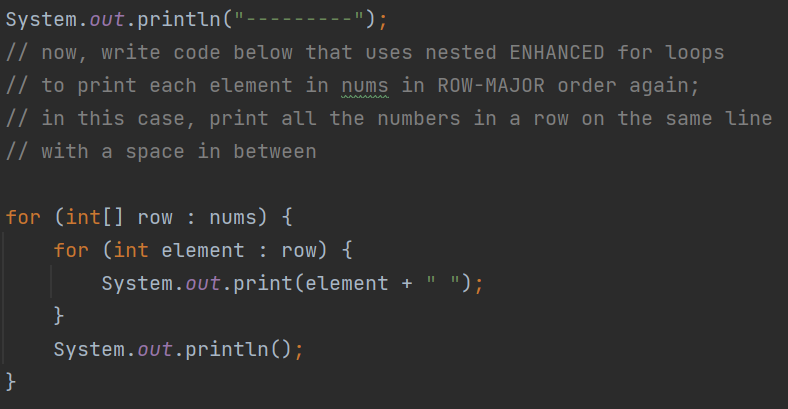
**Done!**

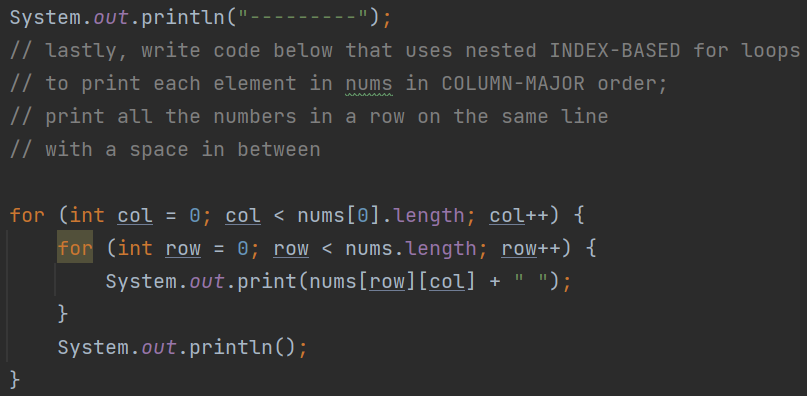
Submit in Google Classroom:

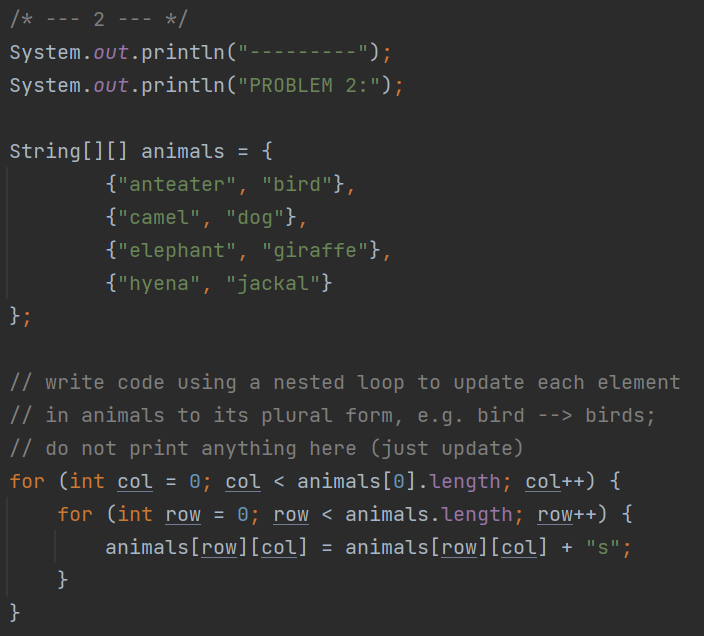


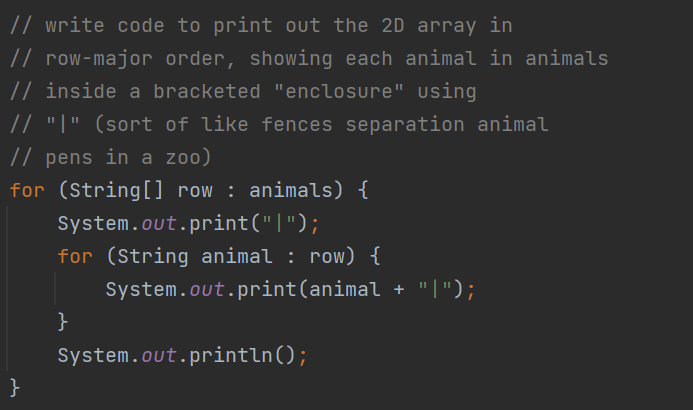
### Solution ([back](#_cqzwapwg8j8g))





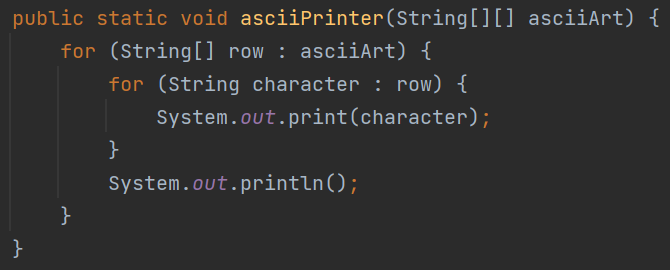


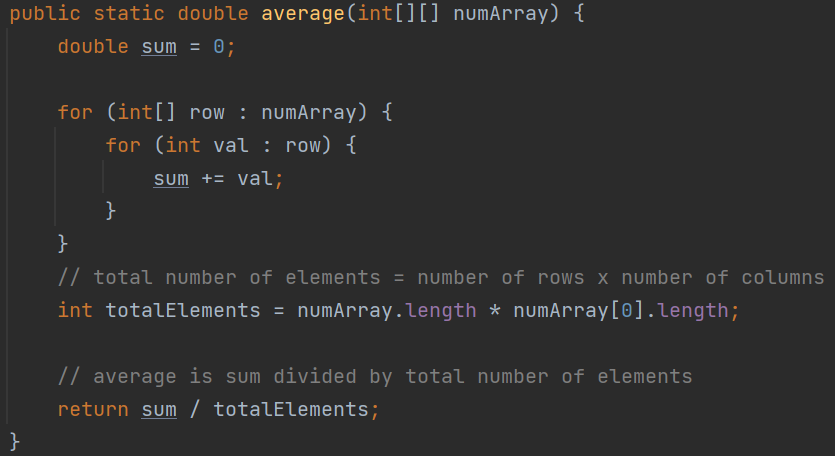


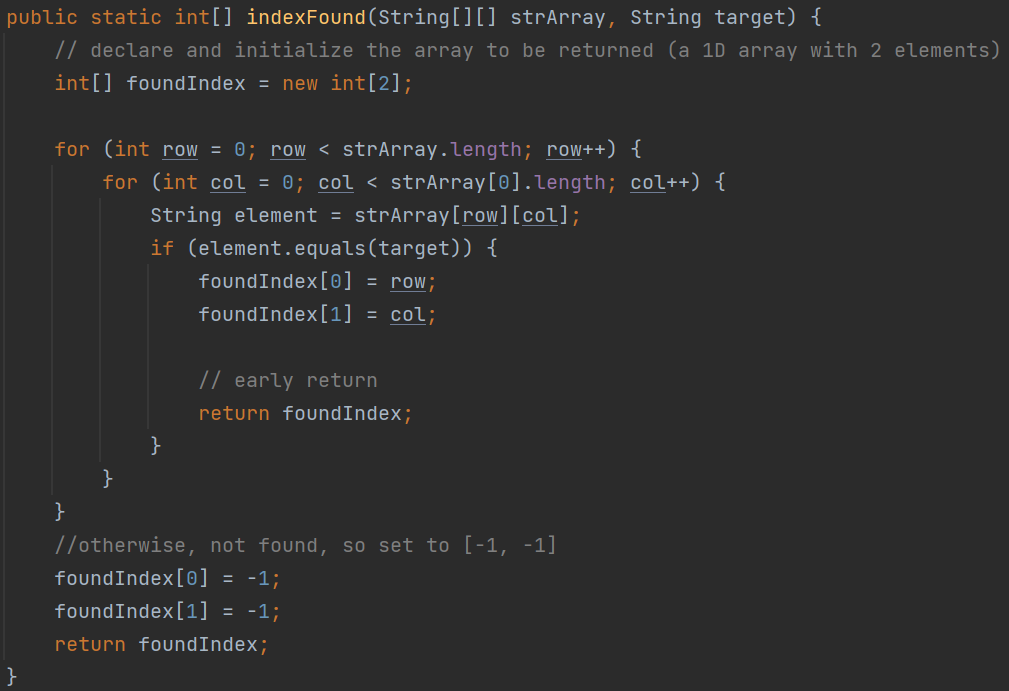


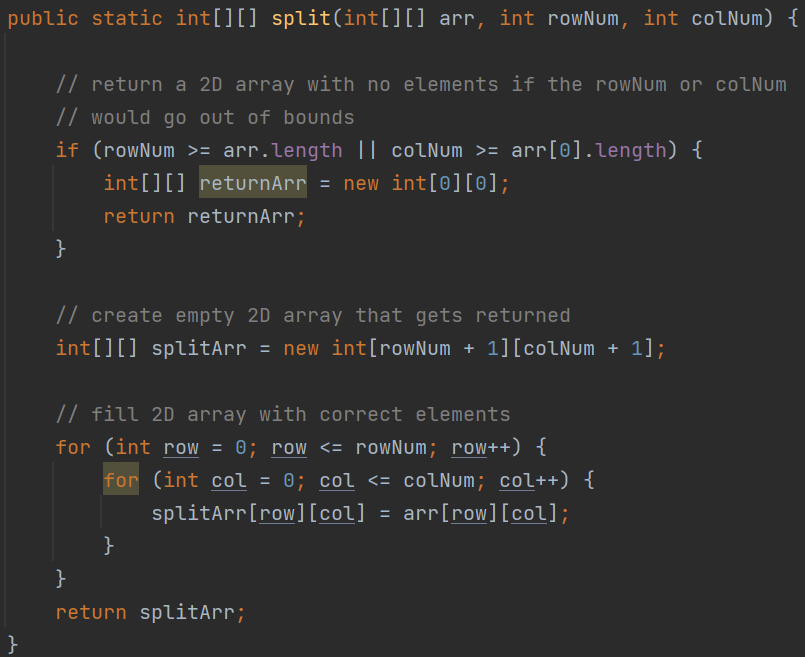
### ([back](#_cqzwapwg8j8g))

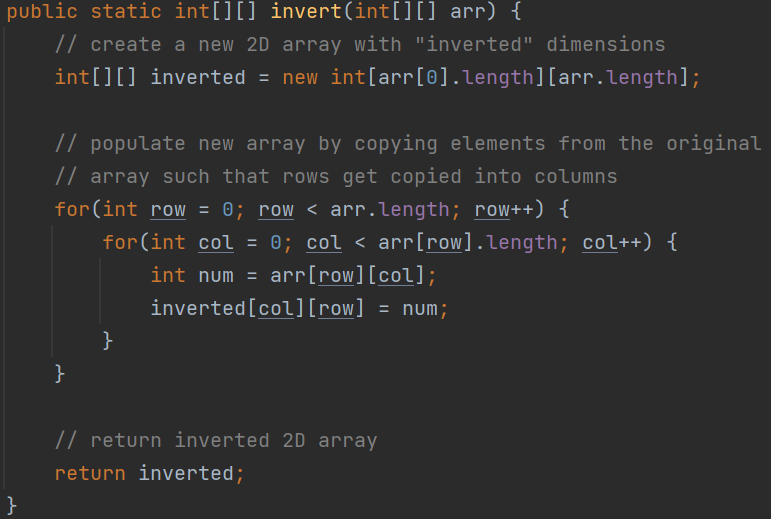
### Solutions ([back](#_8ylqofxafq5u))











### ([back](#_8ylqofxafq5u))