



University of Maryland College Park

Department of Computer Science

CMSC131 Fall 2024

Exam #3

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

KEY

STUDENT ID (e.g. 123456789):

Instructions

- Please print your answers and use a pencil.
- Do not remove the staple from the exam. Removing it will interfere with the Gradescope scanning process.
- To make sure Gradescope can recognize your exam, print your name, write your directory id at the bottom of pages with the text DirectoryId, provide answers in the rectangular areas provided, and do not remove any exam pages. Even if you use the provided extra pages for scratch work, they must be returned with the rest of the exam.
- This exam is a closed-book, closed-notes exam, with a duration of 50 minutes and 100 total points.
- Your code must be efficient.
- Multiple choice questions have only one answer unless indicated otherwise.
- You don't need to use meaningful variable names; however, we expect good indentation.

Grader Use Only

#1	Code 1	16
#2	Code 2	32
#3	Code 3	28
#4	Code 4	24
Total	Total	100

Problem #1 (Code 1)

Finish the static method below. It returns a comma separated **String** (no comma after the last number) of the integers in a diagonal of the 2D array argument. The diagonal is the one from bottom right to top left. The order that elements appear in the returned **String** should also be from bottom right to top left. You can assume a square array where number of rows and columns are the same. Assume the passed in 2D array has 1 or more rows of integers (i.e. no null rows or empty rows). No library methods allowed. Using the **length** field of an array and/or making String variables are allowed. **See main method on the next page.**

```
public static String getDiagonal(int[][] matrix) {

    String diag = "";

    // Traverse the diagonal in reverse order
    for (int i = matrix.length - 1; i >= 0; i--) {
        diag += matrix[i][i];
        if (i > 0) {
            diag += ", ";
        }
    }

    return diag;
}
```

Sample main

```
public static void main(String[] args) {
    int[][] matrix = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    System.out.println("Diagonal: " + getDiagonal(matrix));
}
}
```

Output

Diagonal: 9, 5, 1

Problem #2 (Code 2)

Finish the static method **countVowels**. It returns a 2D array with 5 rows and 2 columns. The first element in each row is a vowel (a,e,i,o,u) and the second is the case-insensitive count of the number of times the vowel is found in the words that appear in the argument array (assume the array has one or more **Strings**, no **nulls**). Notice the array you return is of type **String**, not **char** or **int**. Allowed library methods are:

- `String.valueOf(char c)` - Returns the string representation of the **char** argument.
- `String.valueOf(int i)` - Returns the string representation of the **int** argument.
- `Integer.parseInt(String s)` - Returns the integer value represented by the argument
- Non-static String methods: **toLowerCase**, **length**, and **charAt**
- Making Strings, arrays, and/or using length field of arrays are allowed.

Sample main

```
public static void main(String[] args) {
    String[] words = {"Apple", "Banana", "Kiwi", "Orange", "Mango"};

    // Call the method and print the result
    String[][] result = countVowels(words);

    // Print the result
    for (String[] row : result) {
        System.out.println(row[0] + ": " + row[1]);
    }
}
```

Output

a: 6
e: 2
i: 2
o: 2
u: 0

Write code on the back

```

public static String[][] countVowels(String[] words) {

    // Initialize a 2D array for vowels and counts
    String[][] vowelCounts = new String[5][2];
    char[] vowels = {'a', 'e', 'i', 'o', 'u'};

    // Initialize each row of the 2D array: vowel and count string
    for (int i = 0; i < vowels.length; i++) {
        vowelCounts[i][0] = String.valueOf(vowels[i]);
        vowelCounts[i][1] = "0"; // Initial count for each vowel is 0
    }

    // Loop through each word in the input array
    for (String word : words) {
        // Convert each word to lowercase to make the counting case-insensitive
        word = word.toLowerCase();

        // Count vowels in the current word
        for (int i = 0; i < vowels.length; i++) {
            int count = 0;
            // Count occurrences of each vowel in the current word
            for (int j = 0; j < word.length(); j++) {
                if (word.charAt(j) == vowels[i]) {
                    count++;
                }
            }
            // Add the count to the corresponding vowel's total
            int currentCount = Integer.parseInt(vowelCounts[i][1]);
            currentCount += count;
            vowelCounts[i][1] = String.valueOf(currentCount);
        }
    }

    return vowelCounts;
}

```

Problem #3 (Code 3)

Finish the static method **calculateSum**. It returns the sum of all **Integers** values in the 2D **data** array using the following rules: 1) It will use the **defaultValue** if any cell is **null**, 2) It will negate all **Integers** found in odd rows. You can assume that **data** is a valid rectangular 2D array. A rectangular array has the same number of columns for each row. Assume that the rectangular array has 1 or more rows of **Integers** (i.e. no null rows or empty rows). No library methods allowed. Using the **length** field of an array and/or making **Integer** variables are allowed. Do not modify the array, just return the sum according to rules above.

Sample main

```
public static void main(String[] args) {  
  
    Integer[][] data = { {1, 2, 3},  
                          {4, null, 6},  
                          {7, 8, 9} };  
  
    System.out.println(Matrix.calculateSum(data, 7));  
  
    for (Integer[] row : data) {  
        System.out.println( Arrays.toString(row));  
    }  
  
}
```

Output

```
13  
[1, 2, 3]  
[4, null, 6]  
[7, 8, 9]
```

Write code on the back

```
public static int calculateSum(Integer[][] data, int defaultValue) {

    int sum = 0;

    for (int i = 0; i < data.length; i++) {

        for (int j = 0; j < data[i].length; j++) {
            Integer val = data[i][j];
            if (val == null) {
                val = defaultValue; // Use defaultValue for null
            }

            // If the row is odd, negate the value before adding
            if (i % 2 != 0) {
                val = -val;
            }

            sum += val;
        }
    }
    return sum;
}
```

Problem #4 (Code 4)

Finish the static method **noRepeat**. It returns an **ArrayList<Character>** with only one instance of each character in the 2D array argument **data** (excluding any **null** values). You can assume that **data** is a valid rectangular 2D array with 1 or more rows of Characters (i.e. no null rows or empty rows). **For looping constructs, you can only use enhanced for loops (also known as a for-each loop).** For library methods, you can only use the default **ArrayList** constructor, **add** (the version that adds to the end), and the **contains** methods. Using the **length** field of an array and/or making **Character** variables are allowed (but you really don't need either of them). Do not modify the 2D array, just return the new **ArrayList<Character>**.

Sample main

```
public static void main(String[] args) {  
  
    Character [][]data = new Character[][]{  
        {'a', 'c', 'y'},  
        {'Y', null, 'a'},  
        {'b', 'z', 'z'} };  
  
    ArrayList<Character> result = noRepeat(data);  
    System.out.println("Elements: " + result);  
  
}
```

Output

Elements: [a, c, y, Y, b, z]

Write code on the back

```
public static ArrayList<Character> noRepeat(Character[][] data) {  
  
    ArrayList<Character> uniqueElements = new ArrayList<>();  
    for (Character[] row : data) {  
        for (Character val : row) {  
            if (val != null && !uniqueElements.contains(val)) {  
                uniqueElements.add(val);  
            }  
        }  
    }  
    return uniqueElements;  
}
```