Batch: D3 Roll No.: 16010123294

Experiment / assignment / tutorial No. 04

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Multi-dimensional Arrays (Jagged Array)

AIM: Write a program which stores information about n players in a two dimensional array. The array should contain the number of rows equal to the number of players. Each row will have a number of columns equal to the number of matches played by that player which may vary from player to player. The program should display player number (index +1), runs scored in all matches and its batting average as output. (It is expected to assign columns to each row dynamically after getting value from the user.)

Expected OUTCOME of Experiment:
CO1:Apply the features of object oriented programming languages. (C++ and Java)
CO2:Explore arrays, vectors, classes and objects in C++ and Java

Books/ Journals/ Websites referred:

1. E. Balagurusamy, "Programming with Java", McGraw-Hill.
2. E. Balagurusamy, "Object Oriented Programming with C++", McGraw-Hill.

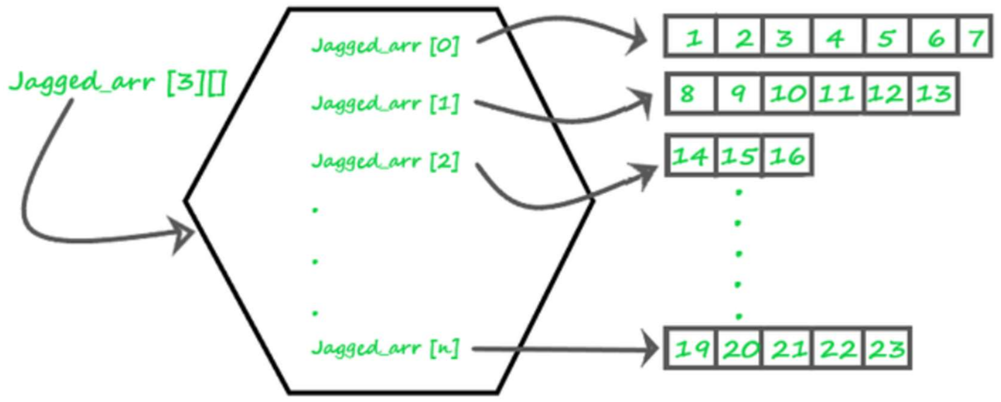
Pre Lab/ Prior Concepts:
Arrays

Multi-Dimensional Array:

```
10  12  43  11  22
20  45  56  1   33
30  67  32  14  44
40  12  87  14  55
50  86  66  13  66
```

60 53 44 12 11
A multi-dimensional array is one that can hold all the values above. You set them up like this:
int[ ][ ] numbers = new int[6][5];
The first set of square brackets is for the rows and the second set of square brackets is for the columns. In the above line of code, we're telling Java to set up an array with 6 rows and 5 columns.
aryNumbers[0][0] = 10;
aryNumbers[0][1] = 12;
aryNumbers[0][2] = 43;
aryNumbers[0][3] = 11;
aryNumbers[0][4] = 22;
So the first row is row 0. The columns then go from 0 to 4, which is 5 items.
Jagged Array:

A jagged array, also known as a "ragged array," is an array of arrays where each "inner" array can have different lengths. This contrasts with a rectangular array (or a multi-dimensional array), where every inner array must have the same length. Jagged arrays are useful when dealing with data structures that naturally vary in size, such as lists of lists or matrices with different numbers of columns.



Theory

1. **Definition**: A jagged array is an array whose elements are arrays, possibly of different lengths. This means that the length of each inner array can vary.
2. **Memory Layout**: Unlike a rectangular array where memory allocation is continuous, each inner array in a jagged array is a separate array stored at different locations in memory.

3. **Usage**: Jagged arrays are often used in scenarios where the data is inherently irregular. For example, they can be useful in representing data structures like adjacency lists in graphs, where different nodes have different numbers of neighbors.
4. **Advantages**:
   ○ **Space Efficiency**: Only the required space is allocated for each sub-array, saving memory when dealing with irregular data.
   ○ **Flexibility**: Allows more flexibility in managing arrays of varying lengths.
5. **Disadvantages**:
   ○ **Complexity**: Increased complexity in managing and accessing elements.
   ○ **Performance**: Potentially lower performance due to non-contiguous memory allocation

Syntax :

```
// Declare a jagged array with 3 elements
 int[][] jaggedArray = new int[3][];
```

Class Diagram:

Algorithm:

Initialize Scanner:

Create a Scanner object to take user input.

Input Number of Players:

Prompt the user to enter the number of players.

Store the input in an integer variable players.

Create Jagged Array:

Create a 2D jagged array runs to store the runs of each player.

Input Runs for Each Player:

Loop through each player (from 0 to players - 1):

Prompt the user to enter the number of matches for the current player.

Initialize the current player's array in runs with the size equal to the number of matches.

For each match, prompt the user to enter the runs scored by the player and store it in the array.

Calculate Average Runs:

Loop through each player (from 0 to players - 1):

Initialize a variable totalRuns to 0.

Sum up the runs for all matches of the current player.

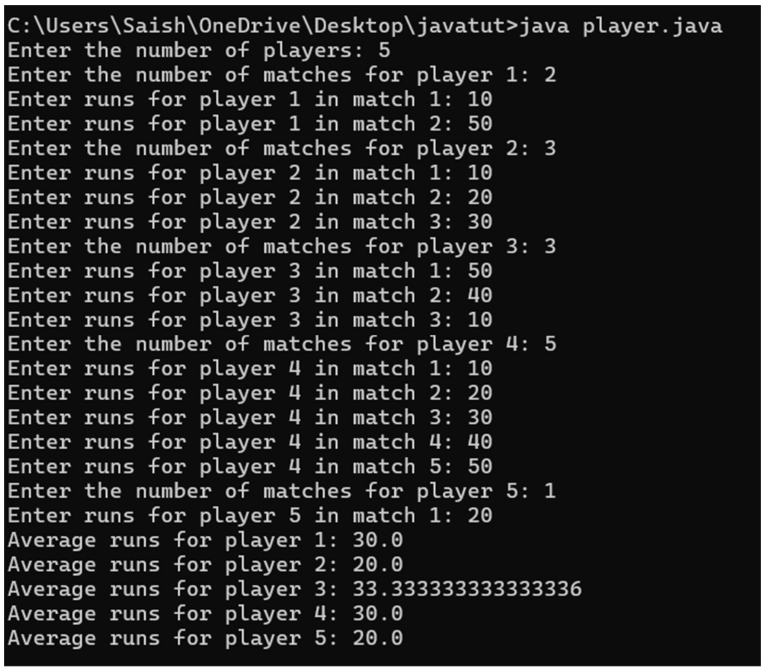
Calculate the average by dividing totalRuns by the number of matches.

Print the average runs for the current player.

Implementation details:

```
import java.util.Scanner;


public class player {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);


        System.out.print("Enter the number of players: ");
        int players = sc.nextInt();


        int[][] runs = new int[players][];
        for (int i = 0; i < players; i++) {
            System.out.print("Enter the number of matches for player " + (i + 1) + ": ");
            int matches = sc.nextInt();
            runs[i] = new int[matches];
            for (int j = 0; j < matches; j++) {
                System.out.print("Enter runs for player " + (i + 1) + " in match " + (j + 1) + ":
");
                runs[i][j] = sc.nextInt();
            }
        }

        for (int i = 0; i < players; i++) {
            int totalRuns = 0;
            for (int j = 0; j < runs[i].length; j++) {
                totalRuns += runs[i][j];
            }
            double average = (double) totalRuns / runs[i].length;
            System.out.println("Average runs for player " + (i + 1) + ": " + average);
        }

        sc.close();
    }
}
```

Output:



```
C:\Users\Saish\OneDrive\Desktop\javatut>java player.java
Enter the number of players: 5
Enter the number of matches for player 1: 2
Enter runs for player 1 in match 1: 10
Enter runs for player 1 in match 2: 50
Enter the number of matches for player 2: 3
Enter runs for player 2 in match 1: 10
Enter runs for player 2 in match 2: 20
Enter runs for player 2 in match 3: 30
Enter the number of matches for player 3: 3
Enter runs for player 3 in match 1: 50
Enter runs for player 3 in match 2: 40
Enter runs for player 3 in match 3: 10
Enter the number of matches for player 4: 5
Enter runs for player 4 in match 1: 10
Enter runs for player 4 in match 2: 20
Enter runs for player 4 in match 3: 30
Enter runs for player 4 in match 4: 40
Enter runs for player 4 in match 5: 50
Enter the number of matches for player 5: 1
Enter runs for player 5 in match 1: 20
Average runs for player 1: 30.0
Average runs for player 2: 20.0
Average runs for player 3: 33.333333333333336
Average runs for player 4: 30.0
Average runs for player 5: 20.0
```

Conclusion:

Learned Implementation of jagged array and also carried out traversal of elements in jagged array with example of average runs of players.

Date: ____________ **Signature of faculty in-charge**

Post Lab Descriptive Questions:

Q.1 Write a program for Given an array arr[] of size N. The task is to find the sum of the contiguous subarray within a arr[] with the largest sum.

```
public class MaxSubarraySum {
    public static int maxSubArraySum(int[] arr) {
        int maxSoFar = arr[0];
        int currentMax = arr[0];

        for (int i = 1; i < arr.length; i++) {
            currentMax = Math.max(arr[i], currentMax + arr[i]);
            maxSoFar = Math.max(maxSoFar, currentMax);
        }

        return maxSoFar;
    }

    public static void main(String[] args) {
        int[] arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
        int maxSum = maxSubArraySum(arr);
        System.out.println("Maximum contiguous subarray sum is: " + maxSum);
    }
}
```

Q.2.Create a jagged array of integers. This array should consist of two 2-D arrays. First 2-D array should contain 3 rows having length of 4,3,and 2 respectively. Second 2-D array should contain 2 rows with length 3 and 4 respectively**.**

```
public class JaggedArrayExample {
    public static void main(String[] args) {
        int[][][] jaggedArray = new int[2][][];

        jaggedArray[0] = new int[3][];
        jaggedArray[0][0] = new int[4];
        jaggedArray[0][1] = new int[3];
        jaggedArray[0][2] = new int[2];

        jaggedArray[1] = new int[2][];
        jaggedArray[1][0] = new int[3];
        jaggedArray[1][1] = new int[4];

        for (int i = 0; i < jaggedArray.length; i++) {
            for (int j = 0; j < jaggedArray[i].length; j++) {
                for (int k = 0; k < jaggedArray[i][j].length; k++) {
                    jaggedArray[i][j][k] = i + j + k;
                }
            }
        }

        for (int[][] array2D : jaggedArray) {
            for (int[] array1D : array2D) {
                for (int num : array1D) {
                    System.out.print(num + " ");
                }
                System.out.println();
            }
            System.out.println();
        }
    }
}
```

Q.3. Consider the following code

int number[] = new int[5];

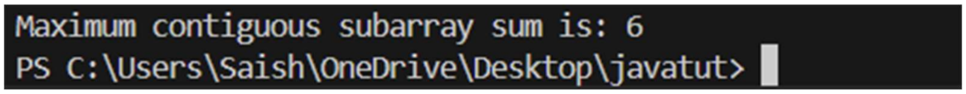
After execution of this statement, which of the following are true?

(A) number[0] is undefined
(B) number[5] is undefined
(C) number[4] is null
(D) number[2] is 0
(E) number.length() is 5

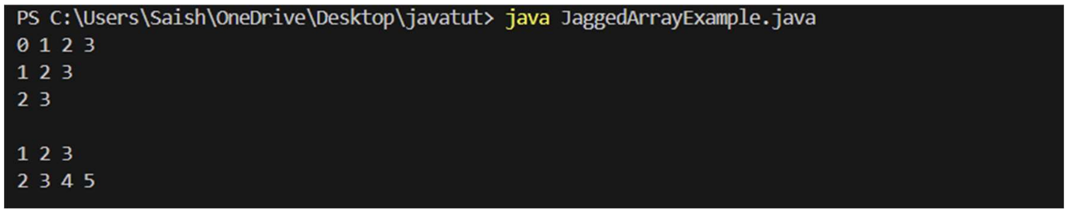
(i) (C) & (E)
(ii) (A) & (E)
(iii) (E)
(iv) (B), (D) & (E)

Ans: iv B,D,E

Output:



```
Maximum contiguous subarray sum is: 6
PS C:\Users\Saish\OneDrive\Desktop\javatut>
```



```
PS C:\Users\Saish\OneDrive\Desktop\javatut> java JaggedArrayExample.java
0 1 2 3
1 2 3
2 3

1 2 3
2 3 4 5
```