INFORMATION & COMPUTER SCIENCE DEPARTMENT

ICS 102 - Introduction to Computing

LAB#14 2-D Arrays

# Objectives:

To gain experience with:

* Declaring, creating and accessing 2D-arrays
* Manipulating 2D-arrays.
* Rugged 2D-arrays.
* Passing and returning 2D-array references to methods.

# 1. Brief Review of 2-D Arrays

**Why 2D-arrays?**

A 2D-array is used to store information that would otherwise require several 1D-arrays to store. For example, assume you have a table of student grades in 5 quizes as follows

**This table can be stored as:**

1. five 1D arrays, each represents a column,
2. six 1D arrays, each represents a row,
3. or one 2D array, in which the position of each element is determined by its row and column

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** |  | **quiz1** | **quiz2** | **quiz3** | **quiz4** | **quiz5** |
| 900000 |  | 50.5 | 40.0 | 60.0 | 0.0 | 55.0 |
| 920000 |  | 70.0 | 60.0 | 75.0 | 90.0 | 66.5 |
| 930520 |  | 65.0 | 70.0 | 65.0 | 80.0 | 78.0 |
| 940000 |  | 80.0 | 90.0 | 95.0 | 85.0 | 100.0 |
| 953478 |  | 40.0 | 30.0 | 50.0 | 55.0 | 45.0 |
| 972893 |  | 60.0 | 50.0 | 39.0 | 70.0 | 55.9 |

The third solution (having a single entity representing 2D array) is more general and convenient for programmers dealing with data that are naturally organized in tables.

**Java 2D-array:**

In Java, a table may be *conceptually* implemented as a 2D array. Each slot of the array is a variable that can hold a value and works like any variable. As with one-dimensional arrays, every slot in a 2D array is of the same type. The type can be a primitive type or an object reference type.

**Note:**

* Rows are numbered from 0 to N-1, where N is the number of rows
* Columns are numbered from 0 to M-1, where M is the number of columns.
* A 2D array with N rows and M columns will have N times M number of slots.
* However, it is possible for a 2D array to have different number of slots in each row.
* The **length** of a 2D array is the number of rows in the array.

**2. Declaring 2D-arrays:**

A 2D-array is an object, and a 2D-array object reference is declared as follows

**type[ ][ ] arrayName;**

**Declaring and creating 2D Array object Example1:**

int[][] grades;

The declaration declares a reference variable **grades** that is expected to hold a reference (i.e., the starting address) to a 2D-array of type int; the value of grades is null.

A 2D array object is created using the **new** operator as follows:

grades = new int[3][5];

creates an array object of **3 rows** and **5 columns**, and puts the reference in grades. All the elements of the array are initialized to zero. Of course, you can combine the declaration and the array instantiation in one statement as follows

int[][] grades = new int[3][5];

You can also achieve the same thing by using an initializer-list as follows:

int[][] grades = { {0,0,0,0,0}, {0,0,0,0,0}, {0,0,0,0,0} };

Using the initializer-list, you can initialize the array elements to any values you want as in the following example:

int[][] myArray = { {8,1,2,2,9}, {1,9,4,0,3}, {0,3,0,0,7} };

creates 2D array of 3 rows and 5 columns and initializes the elements to specified values.

**Declaring and creating 2D Array object Example2:**

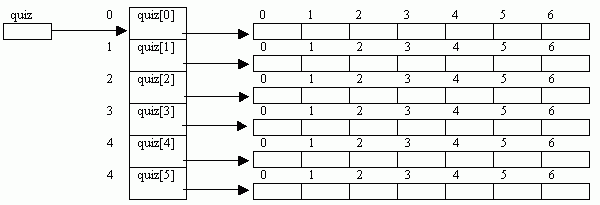
|  |  |
| --- | --- |
| **double[][] array = new double[4][];** // declares a 2D array with 4 rows, the number of  // elements in each row may be the same or it may  // be different | |
| Same number of row elements | Different numberof row elements |
| **array[0] = new double[6];**  **array[1] = new double[6];**  **array[2] = new double[6];**  **array[3] = new double[6];** | **array[0] = new double[2];**  **array[1] = new double[4];**  **array[2] = new double[7];**  **array[3] = new double[5];** |

# Implementation of 2D Array

A two dimensional array is implemented (realized in the memory) as an array of one-dimensional arrays:

# Declaring and creating 2D Array object Example 3:

|  |  |
| --- | --- |
| **double[][] quiz;**  **quiz = new double[6][];**  **quiz[0] = new double[7];**  **quiz[1] = new double[7];**  **quiz[2] = new double[7];**  **quiz[3] = new double[7];**  **quiz[4] = new double[7];**  **quiz[5] = new double[7];** | **double[][] quiz = new double[6][];**  **for(int k = 0; k < quiz.length; k++)**  **quiz[k] = new double[7];** |



It is obvious from the above declarations that each row of a 2-D array, quiz[i] is an independent 1-D array.

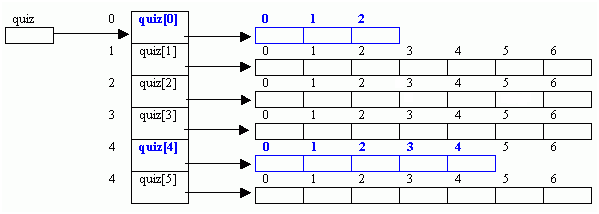
**Declaring rugged 2D-arrays**

Because each row in a 2-D array is an independent 1-D array, it follows that the rows of a 2-D array do not need to be of the same size. For example, if in the declaration of Example 3, we change the declaration of rows 0 and 4 as follows:

**quiz[0] = new double[3];**

**quiz[4] = new double[5];**

Then we have the following **rugged** array.



**3. Accessing individual elements**

An individual element of a 2-D array can be accessed by specifying the index of row and column. For example

grades[2][4] = 95;

assigns the value 95 to the element in the third row and fifth column. The indexed (subscripted) variable can be used wherever an ordinary variable can be used.

## Example 1

The following program fragment prints out the elements of a 2D array row-wise:

int[][] grades={{20, 30, 40, 35},{10, 50, 55, 45},{60, 70, 40, 65}};

for(int i=0; i<3; i++){

for(int j=0; j<4; j++)

System.out.print(grades[i][j]+”\t”);

System.out.println();

}

the output is:

20 30 40 35

10 50 55 45

60 70 40 65

# Individual Rows of a 2D-array can be replaced

You can change any row in the array as in the following example, it creates a new 1D array then assign its reference to the first reference in the 2D array myArray:

int[] x = {1, 9, 4} ; // declare and inititializes x

myArray[0] = x ; // assign to myArray

Notice that the following assignment is wrong:

myArray[0] = {1, 9, 4} ; //Wrong

An initializer list can only be used to initialize an array, not to assign values to it during a run of a program.

# 3. 2D array manipulation

Example: Each of 4 students has taken 3 quizzes

double[][] quiz = {{2.0, 1.5, 1.7}, {0.5, 1.0, 0.0}, {2.0, 2.0, 2.0}, {1.2, 0.7, 1.0}};

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 2.0 | 1.5 | 1.7 |
| 1 | 0.5 | 1.0 | 0.0 |
| 2 | 2.0 | 2.0 | 2.0 |
| 3 | 1.2 | 0.7 | 1.0 |

* 1. To manipulate a single row, fix the row index and use a single column loop

Example: Find the total quiz score for student with row index 3

**int c, r;**

**double sum = 0;**

**for(c = 0; c < quiz[3].length; c++)**

**sum += quiz[3][c];**

**System.out.printf(“Total quiz score for student#3 = %.2f\n”, sum);**

* 1. To manipulate a single column, fix the column index and use a single row loop

Example: Find the average of quiz#0

**int c, r;**

**double sum = 0;**

**for(r = 0; r < quiz.length; r++)**

**sum += quiz[r][0];**

**System.out.printf(“Quiz#0 average = %.2f\n”, sum / 4);**

* 1. To manipulate each row, use nested loops in which the row loop is the outer loop

Example: Find the total quiz score for each student

**int c, r;**

**double sum;**

**for(r = 0; r < quiz.length; r++){**

**sum = 0;**

**for(c = 0; c < quiz[0].length; c++){**

**sum += quiz[r][c];**

**}**

**System.out.printf(“Total quiz score for student#%d = %.2f\n”, r, sum);**

**}**

Example: Read the quiz array row-wise:

**int c, r;**

**double [][] quiz = new double[4][3];**

**for( r = 0; r < quiz.length; r++ ){**

**for(c = 0; c < quiz[0].length; c++){**

**quiz[r][c] = scanner.nextDouble();**

**}**

**}**

**Suppose the input is:**

**5.0 2.0**

**3.0**

**4.0 1.0 3.5 2.5**

**1.5 6.0 5.5 7.0**

**8.0**

**The array quiz is initialized as:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 5.0 | 2.0 | 3.0 |
| 1 | 4.0 | 1.0 | 3.5 |
| 2 | 2.5 | 1.5 | 6.0 |
| 3 | 5.5 | 7.0 | 8.0 |

* 1. To manipulate each column, use nested loops in which the column loop is the outer loop

Example 01: Find the average of each quiz

**int c, r;**

**double sum;**

**for( c = 0; c < quiz[0].length; c++ ){**

**sum = 0;**

**for( r = 0; r < quiz.length; r++ ){**

**sum += quiz[r][c];**

**}**

**System.out.printf(“Quiz#%d average = %.2f\n”, c, sum / 4);**

**}**

Example 02: Read the quiz array column-wise:

**int c, r;**

**double[][] quiz = new double[4][3];**

**for(c = 0; c < quiz[0].length; c++){**

**for(r = 0; r < quiz.length; r++){**

**quiz[r][c] = scanner.nextDouble();**

**}**

**}**

**Suppose the input is:**

**5.0 2.0**

**3.0**

**4.0 1.0 3.5 2.5**

**1.5 6.0 5.5 7.0**

**8.0**

**The array quiz is initialized as:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 5.0 | 1.0 | 6.0 |
| 1 | 2.0 | 3.5 | 5.5 |
| 2 | 3.0 | 2.5 | 7.0 |
| 3 | 4.0 | 1.5 | 8.0 |

Example 03: Initialize a ragged 2D array column-wise:

**double[][] x = new double[5][];**

**x[0] = new double[2];**

**x[1] = new double[4];**

**x[2] = new double[1];**

**x[3] = new double[3];**

**x[4] = new double[2];**

**int c = 0,r = 0, count = 0, k = 0;**

**for(c = 0; c < 4; c++){ // 4 is the max number of columns**

**System.out.printf("Column %d: ",k++);**

**for(r= 0; r < x.length; r++){**

**if(c < x[r].length){**

**x[r][c] = count++;**

**System.out.print(x[r][c]+ " ");**

**}**

**}**

**System.out.println();**

**}**

The array **x** is initialized as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | **0** | **5** |  |  |
| 1 | **1** | **6** | **9** | **11** |
| 2 | **2** |  |  |  |
| 3 | **3** | **7** | **10** |  |
| 4 | **4** | **8** |  |  |

Note: There are problems that can be solved by manipulating a 2D-array either row-wise or column-wise:

Examples:

* Find a maximum or minimum element.
* Find the sum or product of all elements in the array

**int c, r;**

**double sum = 0;**

**for( r = 0; r < quiz.length; r++ ){**

**for(c = 0; c < quiz[0].length; c++){**

**sum += quiz[r][c];**

**}**

**}**

**System.out.printf("Sum of all quiz grades = %.2f\n", sum);**

or:

**int c, r;**

**double sum = 0;**

**for(c = 0; c < quiz[0].length; c++){**

**for(r = 0; r < quiz.length; r++){**

**sum += quiz[r][c];**

**}**

**}**

**System.out.printf("Sum of all quiz grades = %.2f\n", sum);**

**4. Passing 2D array references to methods**

A 2D-array is passed to a method by using the array reference variable as an argument, the

corresponding parameter is declared as a reference to a two-dimensional array.

Example:

**import java.util.Scanner;**

**import java.util.InputMismatchException;**

**public class Sum2DArrays2 {**

**public static void main(String[] args) {**

**try{**

**int[][] matrix1, matrix2, matrix3;**

**matrix1 = readArray( );**

**matrix2 = readArray( );**

**matrix3 = addMatrix(matrix1, matrix2);**

**System.out.printf("%nThe sum of matrix1 and matrix2 is:%n");**

**displayArray(matrix3);**

**}catch(InputMismatchException | IllegalStateException |**

**IllegalArgumentException e){**

**System.out.println(e);**

**}**

**}**

**private static void displayArray(int[][] x){**

**// Display matrix x row-wise**

**for(int r = 0; r < x.length; r++){**

**for(int c = 0; c < x[r].length; c++){**

**System.out.printf("%3d ", x[r][c]);**

**}**

**System.out.printf("%n");**

**}**

**}**

**private static int[][] readArray( ){**

**Scanner scanner = new Scanner(System.in);**

**System.out.printf("Enter number of rows: ");**

**int numRows = scanner.nextInt();**

**System.out.printf("Enter number of columns: ");**

**int numColumns = scanner.nextInt();**

**if(numRows <= 0 || numColumns <= 0){**

**throw new IllegalStateException("Invalid number of rows or columns");**

**}**

**int[][] x = new int[numRows][numColumns];**

**System.out.printf("Enter %d \* %d elements of matrix row-wise:%n", numRows, numColumns);**

**for(int r = 0; r < x.length; r++){**

**for(int c = 0; c < x[r].length; c++){**

**x[r][c] = scanner.nextInt();**

**}**

**}**

**return x;**

**}**

**private static int[][] addMatrix(int[][] x, int[][] y){**

**if(x.length != y.length || x[0].length != y[0].length)**

**throw new IllegalArgumentException("Array dimensions are not equal");**

**int[][] m = new int[x.length][x[0].length];**

**for(int r = 0; r < x.length; r++){**

**for(int c = 0; c < y[r].length; c++){**

**m[r][c] = x[r][c] + y[r][c];**

**}**

**}**

**return m;**

**}**

**}**

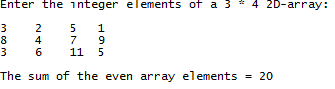
Laboratory Tasks

1.Write a Java program that prompts for and reads a 3\*4 integer array **row-wise**. It then computes and

displays the sum of the even integers in the array. Use appropriate loops and only the **main** method

in your solution.

Sample program run:



2. Write a private static method **linearSearch2D** that takes a 2D integer array and an integer value to

be searched in the array. The method returns the row and column indexes of the first element in the

array that equals to the search value (Note: search the array **column-wise**). If the search is not

successful, the method returns -1 and -1 for the row and column index [Use a two element integer

array to return the two indexes].

Write a main method to test the **linearSearch2D** method. Use the dimensions 4 \* 3 for your

2D-array.

Sample program runs:

|  |
| --- |
|  |
|  |
|  |

3. The correct answers to a 5 question true/false test are: T T F F T. Write a Java program that

initializes a 1D **char** array with the correct answers, and a 2D **char** array of size 6 \* 5 with the

following 6 student answers:

T F T T T

T T T T T

T T F F T

F T F F F

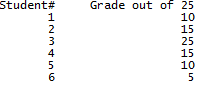
F F F F F

F T T T F

The program then passes these two arrays to a method **getGrades** that returns the grades of the six

students in a 1D integer array. The main method finally displays the grades of the six students.

Assuming that each question is worth 5 points, the output, in the **main** method, is:



4. Each line of a text-file **grades.txt** contains a student ID and his grades in **5** quizzes:

|  |
| --- |
| 9001230 80.0 90.0 70.5 100.0 60.0  9001232 98.0 85.0 100.0 99.0 89.0  9001234 90.0 72.0 0.0 78.0 98.0  9001236 85.0 72.5 95.0 75.0 64.5  9001238 67.0 11.0 28.0 89.5 85.0 |

Given that the number of students is **5**, write a Java program that reads the data from **grades.txt** into an integer 1D-array of student IDs and the corresponding quiz grades in a parallel 2D-array of type **double**. The program then calls the method **studentAverages** that receives the 2D array of grades and returns a 1D-array containing the average of each student. The main method finally prints an output in the format:

**ID Quiz Average**

9001230 80.1

9001232 94.2

9001234 67.6

9001236 78.4

9001238 56.1

Note:

* Your program must use appropriate loops.
* Your program must be general; it must work for any text-file with the above format.
* The **studentAverages** method must not contain **input** and **output** statements.