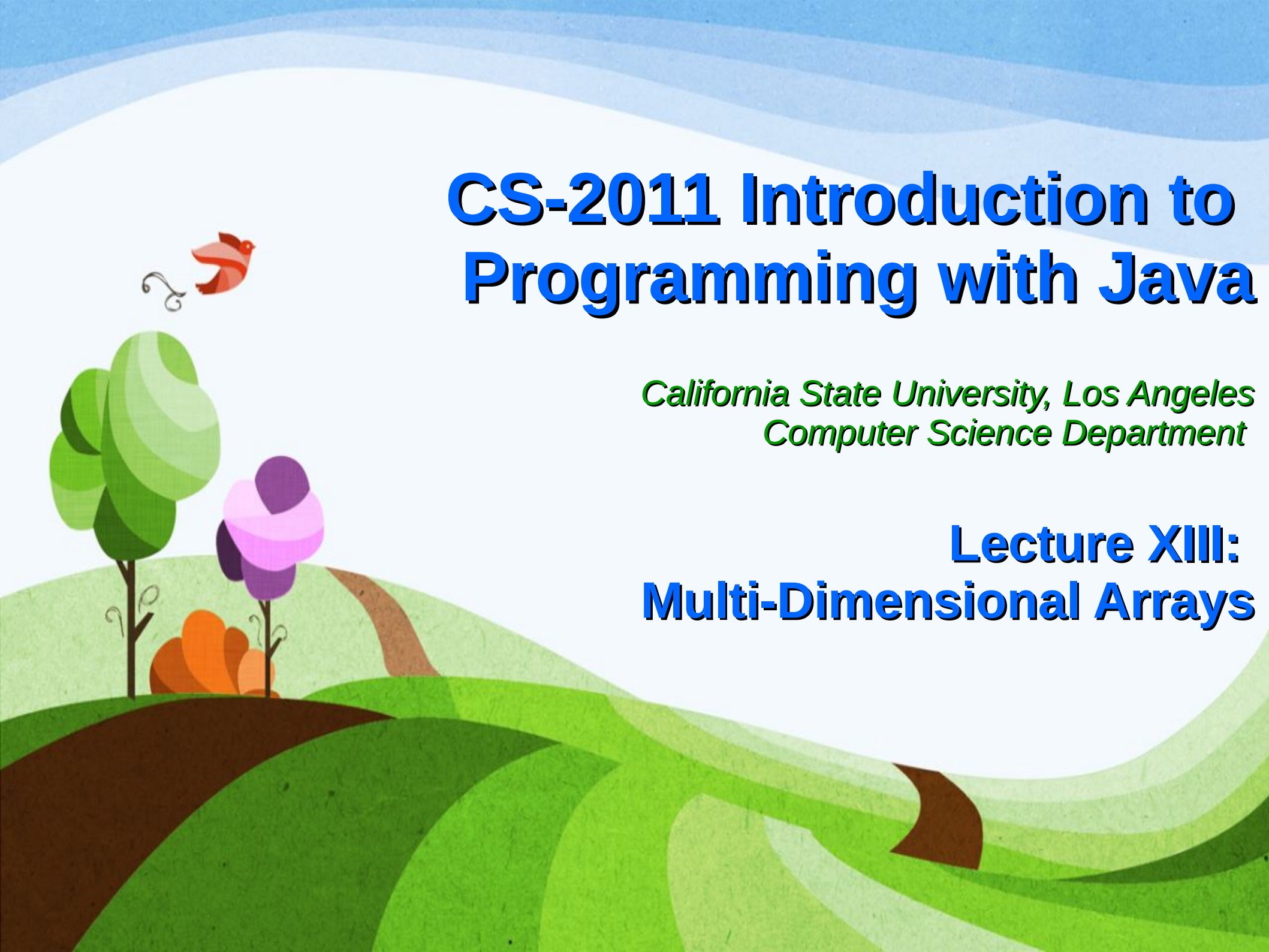


CS-2011 Introduction to Programming with Java

California State University, Los Angeles
Computer Science Department

Lecture XIII: Multi-Dimensional Arrays


Multi-Dimensional Arrays

Introduction

- So far, we have seen how to store a set of data using an array
  - Only has one dimension
  - Used to model linear collections of elements
- You can use a two-dimensional array to represent a matrix or a table.
  - We can even have arrays in greater dimensions

Introduction

- Example: The following table lists the distances between cities. We can store this using a two-dimensional array.

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

Introduction

```
double[][] distances = {
    {0, 983, 787, 714, 1375, 967, 1087},
    {983, 0, 214, 1102, 1763, 1723, 1842},
    {787, 214, 0, 888, 1549, 1548, 1627},
    {714, 1102, 888, 0, 661, 781, 810},
    {1375, 1763, 1549, 661, 0, 1426, 1187},
    {967, 1723, 1548, 781, 1426, 0, 239},
    {1087, 1842, 1627, 810, 1187, 239, 0},
};
```

What exactly is a 2D Array?

- A 2D array is an array of arrays.
- It is an array where each element of the array is another array.
- You can also think of it like an array of reference variables to other arrays.
- Also realize each “row” of the array is an array itself


Two-Dimensional Array Basics

Declaring a 2D Array Reference Variable

- Syntax:

```
dataType[][] refVar;
```

- Example:

```
double[][] table1;
int[][] table2;
String[][] table3;
```

- NOTE: Like 1D arrays this **DOES NOT** create the actual array in memory yet. This is simply a reference type variable which will hold a **reference** to a 2D array later on.

Creating the Array in Memory

- Syntax:

```
dataType[][] refVar;
refVar = new dataType[rows][cols];
```

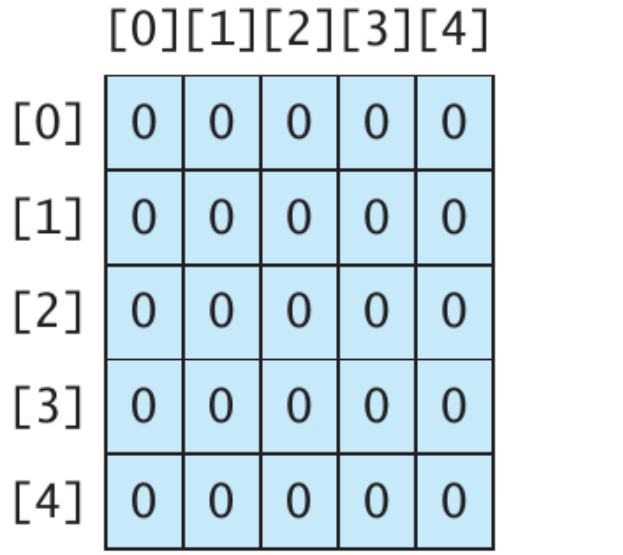
- Can declare the reference and create the array all on one line.

```
dataType[][] refVar = new dataType[rows][cols];
```

- Example

```
int[][] table1 = new int[2][3]; //2 rows 3 cols
double[][] table2 = new double[7][7]; 7 rows 7 cols
```

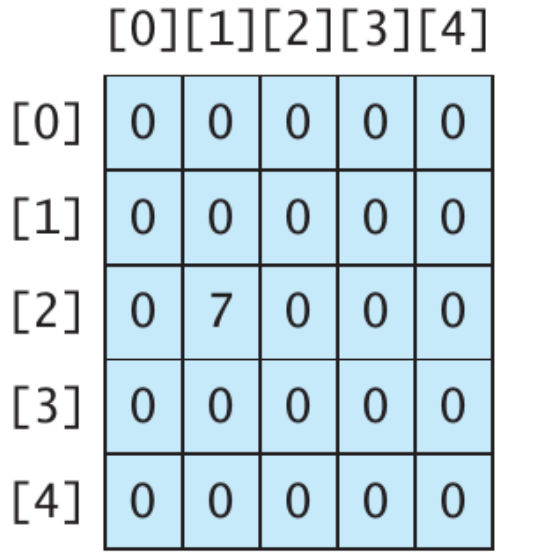
Examples



	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

```
matrix = new int[5][5];
```

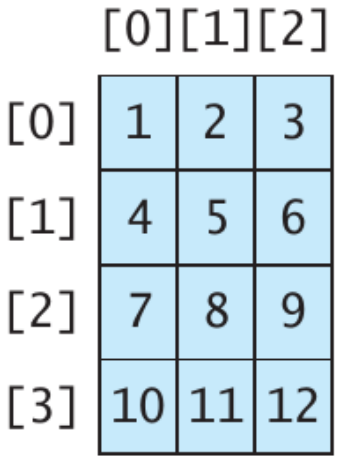
(a)



	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	7	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

```
matrix[2][1] = 7;
```

(b)



	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9
[3]	10	11	12

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

(c)

Initializer Lists

- Just like with 1D arrays 2D arrays can use an initializer list.

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

(a)

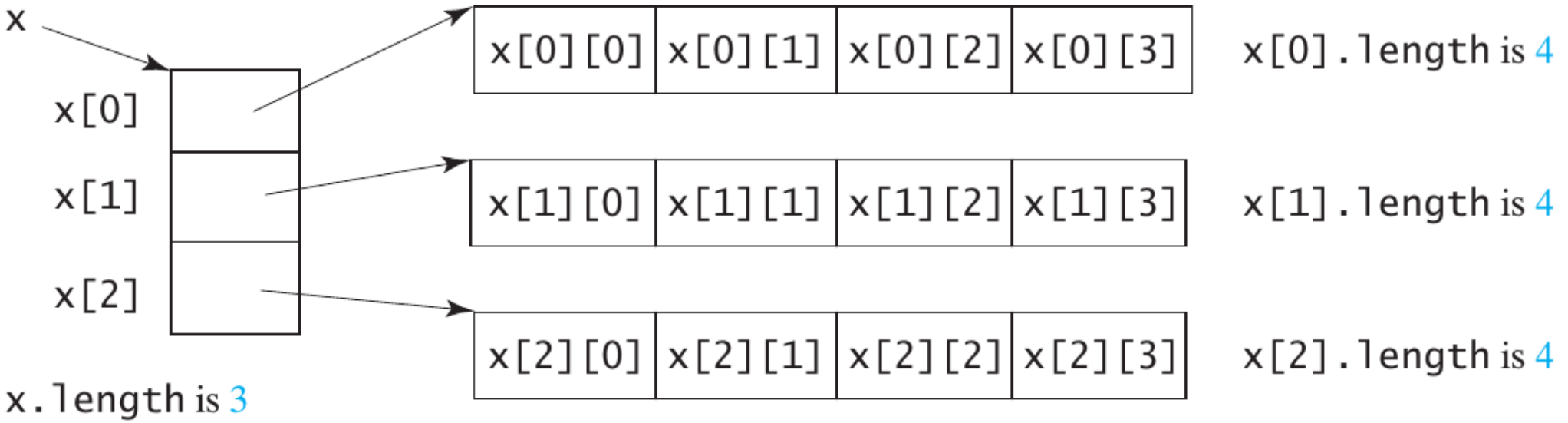
Equivalent

```
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

(b)

Lengths of 2D Arrays

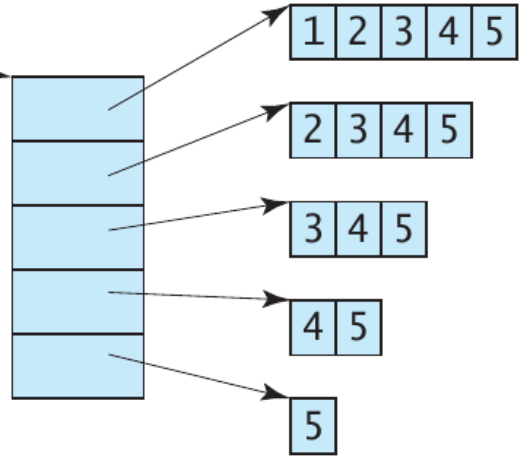
- Remember: A 2D array is an array in which each element is a one-dimensional array.



Ragged Arrays

- Since each row in a two-dimensional array is also an array, the rows can have different lengths.
- This is known as a ***ragged array***.

```
int[][] triangleArray = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```



- What are the lengths of the following?
  - **`triangleArray.length = ??`**
  - **`triangleArray[0].length = ?? triangleArray[1].length = ??`**
  - **`triangleArray[2].length = ?? triangleArray[3].length = ??`**
  - **`triangleArray[4].length = ??`**

Ragged Arrays

If you don't know the values in a ragged array in advance, but do know the sizes—say, the same as before—you can create a ragged array using the following syntax:

```
int[][] triangleArray = new int[5][];
triangleArray[0] = new int[5];
triangleArray[1] = new int[4];
triangleArray[2] = new int[3];
triangleArray[3] = new int[2];
triangleArray[4] = new int[1];
```

You can now assign values to the array. For example,

```
triangleArray[0][3] = 50;
triangleArray[4][0] = 45;
```

Processing Two-Dimensional Arrays

Processing Two-Dimensional Arrays

- Since 2D arrays have more than one dimension, you need to use nested for loops to process them.

- Generally speaking, the more dimensions an array has, the more levels of nesting are required to process the data in the array.

- For the following examples assume:

```
int[][] matrix = new int[10][10];
```

Initializing the Array with User Input

1. *Initializing arrays with input values.* The following loop initializes the array with user input values:

```
java.util.Scanner input = new Scanner(System.in);
System.out.println("Enter " + matrix.length + " rows and " +
  matrix[0].length + " columns: ");
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = input.nextInt();
  }
}
```

Initializing Arrays with Random Value

2. *Initializing arrays with random values.* The following loop initializes the array with random values between 0 and 99:

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = (int)(Math.random() * 100);
  }
}
```

Printing a 2D Array

3. *Printing arrays.* To print a two-dimensional array, you have to print each element in the array using a loop like the following:

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    System.out.print(matrix[row][column] + " ");
  }

  System.out.println();
}
```

Summing All Elements

4. *Summing all elements.* Use a variable named `total` to store the sum. Initially `total` is `0`. Add each element in the array to `total` using a loop like this:

```
int total = 0;
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    total += matrix[row][column];
  }
}
```

Summing All Elements by Column

5. *Summing elements by column.* For each column, use a variable named **total** to store its sum. Add each element in the column to **total** using a loop like this:

```
for (int column = 0; column < matrix[0].length; column++) {
  int total = 0;
  for (int row = 0; row < matrix.length; row++)
    total += matrix[row][column];
  System.out.println("Sum for column " + column + " is "
    + total);
}
```

Which Row has the Largest Sum?

6. *Which row has the largest sum?* Use variables **maxRow** and **indexOfMaxRow** to track the largest sum and index of the row. For each row, compute its sum and update **maxRow** and **indexOfMaxRow** if the new sum is greater.

```
int maxRow = 0;
int indexOfMaxRow = 0;

// Get sum of the first row in maxRow
for (int column = 0; column < matrix[0].length; column++) {
  maxRow += matrix[0][column];
}

for (int row = 1; row < matrix.length; row++) {
  int totalOfThisRow = 0;
  for (int column = 0; column < matrix[row].length; column++)
    totalOfThisRow += matrix[row][column];

  if (totalOfThisRow > maxRow) {
    maxRow = totalOfThisRow;
    indexOfMaxRow = row;
  }
}

System.out.println("Row " + indexOfMaxRow
  + " has the maximum sum of " + maxRow);
```

Random Shuffling

7. *Random shuffling*. Shuffling the elements in a one-dimensional array was introduced in Section 7.2.6. How do you shuffle all the elements in a two-dimensional array? To accomplish this, for each element `matrix[i][j]`, randomly generate indices `i1` and `j1` and swap `matrix[i][j]` with `matrix[i1][j1]`, as follows:

```
for (int i = 0; i < matrix.length; i++) {
  for (int j = 0; j < matrix[i].length; j++) {
    int i1 = (int)(Math.random() * matrix.length);
    int j1 = (int)(Math.random() * matrix[i].length);

    // Swap matrix[i][j] with matrix[i1][j1]
    int temp = matrix[i][j];
    matrix[i][j] = matrix[i1][j1];
    matrix[i1][j1] = temp;
  }
}
```


Passing Two-Dimensional
Arrays to Methods

Passing Two-Dimensional Arrays to Methods

- Same as when you pass a 1D array to a method, but you need two sets of `[]` instead of one.
- Example
  - You could have a method header like this:

```
public static int sum(int[][] m)
```

- You can also return a 2D array from a method.
- Example
  - You would have a method header like this

```
public static int[][] initArray()
```

- See Code: **`PassTwoDimensionalArray.java`**


Case Study: Grading a
Multiple-Choice Test

Case Study: Grading a Multiple-Choice Test

- Assume there are eight students and ten questions.
- The answers are stored in a two dimensional array.
- Each row of the array records a student's answers to the questions.
- The answer key is stored in a 1D array.
- See Code: **`GradeExam.java`**

Case Study: Grading a Multiple-Choice Test

Students' Answers to the Questions:

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

The key is stored in a one-dimensional array:

Key to the Questions:

	0	1	2	3	4	5	6	7	8	9
Key	D	B	D	C	C	D	A	E	A	D


Multi-Dimensional Arrays

Multi-Dimensional Arrays

- Occasionally, you will need to represent n-dimensional data structures.
- In Java, you can create n-dimensional arrays for any integer n.
- The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays for n >= 3.
- For example, the following syntax declares a three-dimensional array variable scores, creates an array, and assigns its reference to scores.
- `double[][][] scores = new double[6][5][2];`

Multi-Dimensional Arrays

- Example: write a program that calculates the total score for students in a class.
- Suppose the scores are stored in a three-dimensional array named **`scores`**.
- What the indexes mean:
  - the first index refers to a student
  - the second index refers to an exam
  - the third index refers to the part of the exam.
- Suppose there are 6 students, 5 exams, and each exam has 2 parts--the multiple-choice part and the programming part.

Multi-Dimensional Arrays

- See Code: MultiArray.java