Multidimensional Arrays

CSE 114: Introduction to Object-Oriented Programming

Paul Fodor

Stony Brook University

http://www.cs.stonybrook.edu/~cse114

Contents

- Two-dimensional arrays
- Declare and Create Two-dimensional Arrays
- Default Values
- lengths and Indexed Variables
- Initializing Using Shorthand Notations
- Two-dimensional Arrays Storage
- Ragged Arrays
- Algorithms: Initializing 2D arrays, Printing, Summing Shuffling
- N-dimensional Arrays

Multidimensional Arrays

- A two-dimensional array to represent a matrix or a table
  - Example: the following table that describes the distances between the cities can be represented 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

Declare Two-dimensional Arrays

```
// Declare array ref var
elementDataType[] [] refVar;

// Alternative syntax - Not preferred!
elementDataType refVar[][];
elementDataType[] refVar[];
```

Create Two-dimensional Arrays

```
// Create array and assign its reference to variable
refVar = new elementDataType[N][M];
```

- **Example: a matrix of 5 rows of 10 elements each:**

```
int[][] matrix;
matrix = new int[5][10];
```

Declare/Create Two-dimensional Arrays

```
// Combine declaration and creation in one statement
elementDataType[][] refVar =
  new elementDataType[N][M];
```

- **Example: a matrix of 5 rows of 10 elements each:**

```
int[][] matrix = new int[5][10];
```

Indexed Variables

- **Indexed variables: 0-based index for first dimension first (row), then second dimension (column) second**

- **Examples:**

```
int[][] matrix = new int[5][10];
matrix[0][0] = 3;
matrix[0][1] = matrix[0][0] + 3;
```

Default Values

- **Default values:**
  - numeric types get 0
  - boolean get false
  - char gets '\u0000'
  - reference types (e.g., String): null

Examples:

```
int[][] matrix = new int[5][10];
System.out.print(matrix[0][0]); // 0
System.out.print(matrix[0][1]); // 0
System.out.print(matrix[0][2]); // 0
```

Declaring, Creating, and Initializing Using Shorthand Notations

You can also use an array initializer to declare, create and initialize a two-dimensional array. For example,

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

Same as

```
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;
```

Alternative syntax:

```
int[][] array = new int[3][];
array[0] = new int[4];
array[1] = new int[4];
array[2] = new int[4];
array[0][0] = 1;...
```

Lengths of a 2-dimensional array

- `matrix.length` is the size of the first dimension (number of rows)

- `matrix[i].length` is the size of the row `i` (i.e., the number of columns in row i)

Lengths of Two-dimensional Arrays

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

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

```
array.length      4
array[0].length 3
array[1].length 3
array[2].length 3
array[3].length 3
```

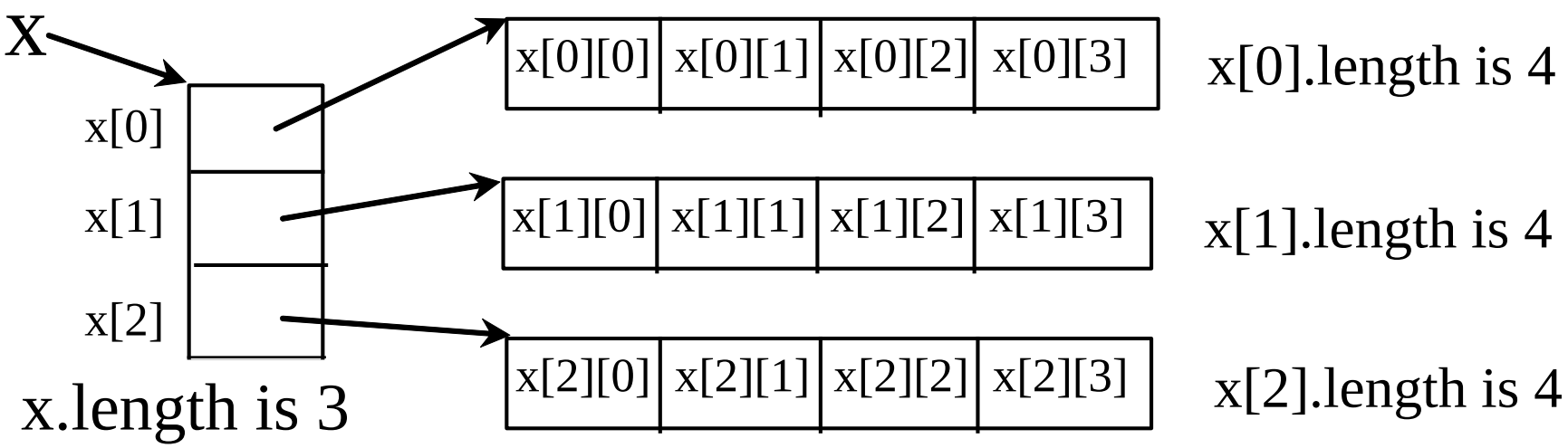
```
array[4].length
```

ArrayIndexOutOfBoundsException

Two-dimensional Arrays

Storage

```
int[][] x = new int[3][4];
```



Ragged Arrays

- A ***ragged array*** is an array where at least 2 rows have different lengths:

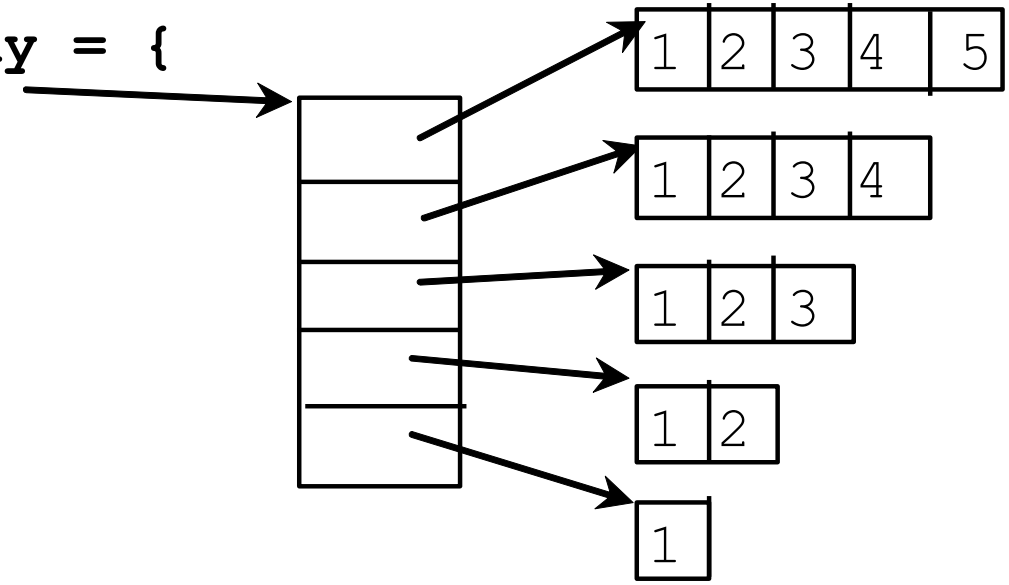
```
int[][] matrix = {
   {1, 2, 3, 4, 5},
   {2, 3, 4, 5},
   {3, 4, 5, 6},
   {4, 5},
   {5, 4, 3, 2, 1}
};
```

```
matrix.length is 5
matrix[0].length is 5
matrix[1].length is 4
matrix[2].length is 4
matrix[3].length is 2
matrix[4].length is 5
```

Ragged Arrays

Storing a ragged array:

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



Initializing 2D arrays with input values

```
int matrix[][] = new int[5][10];
java.util.Scanner input =
   new java.util.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 2D arrays with random values

- **Initialize matrix with random values:**

```
for (int i = 0; i < matrix.length; i++)
  for (int j = 0; j < matrix[i].length; j++)
    matrix[i][j] = (int)(Math.random() * 1000);
```

Printing 2D arrays

```
for(int row=0; row<matrix.length; row++){
    for(int column = 0; column<matrix[row].length;
      column++){
     System.out.print(matrix[row][column] + " ");
    }
    // new line after each row
    System.out.println();
}
```

Printing 2D arrays with for-each

```
for(int[] row:matrix) {
    for(int elem:row){
            System.out.print(elem + " ");
    }
    // new line after each row
    System.out.println();
}
```

Summing all elements

```
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 with for-each

```
int total = 0;
for(int[] row:matrix){
    for(int elem:row){
            total += elem;
    }
}
```

Summing elements by column

```
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);
}
```

2D Random shuffling

```
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
       int i2 = (int)(Math.random() * matrix.length);
       int j2 = (int)(Math.random() * matrix[i2].length);
       // Swap matrix[i][j] with matrix[i2][j2]
       int temp = matrix[i][j];
       matrix[i][j] = matrix[i2][j2];
       matrix[i2][j2] = temp;
    }
}
```

Check Ragged

```
public class Test {
  public static boolean isRagged(int[][] m){
    int l0 = m[0].length;
    for(int i=1; i<m.length; i++)
      if(m[i].length != l0)
        return true;
    return false;
  }
  public static void main(String[] args) {
    int[][] a = {
        {1,2,3},
        {1,2},
        {1,2,3}
      };
    System.out.println(isRagged(a)); // true
  }}
```

Check Ragged with for-each

```
public class Test {
  public static boolean isRagged(int[][] m){
    int l0 = m[0].length;
    for(int[] row)
      if(row.length != l0)
        return true;
    return false;
  }
  public static void main(String[] args) {
    int[][] a = {
        {1,2,3},
        {1,2},
        {1,2,3}
      };
    System.out.println(isRagged(a)); // true
  }}
```

N-dimensional Arrays

- Not just only 2-dimensional arrays!!!
- N-dimensional data structures example:

- grades on multiple dimensions (2 courses, 100 students each course, 25 lab grades):

```
int[][][] scores = new int[2][100][25];

scores[0][0][0] = 3;
```

N-dimensional Arrays

- Printing N-dimensional Arrays:

```
int[][][] m = new int[2][3][4];
for(int i=0; i<m.length; i++) {
  for(int j=0; j<m[i].length; j++) {
    for(int k=0; k<m[i][j].length; k++)
      System.out.print(m[i][j][k] + " ");
    System.out.println();
  }
  System.out.println();
}
```

N-dimensional Arrays

- Printing N-dimensional Arrays with for-each:

```
int[][] [] m = new int[2][3][4];
for(int[] [] slice:m) {
   for(int[] row:slice) {
     for(int e:row)
       System.out.print(e + " ");
     System.out.println();
   }
   System.out.println("\n");
}
```