COMP30820
Java Programming (Conv)

Michael O’Mahony

Chapter 8 Multidimensional Arrays

Motivations

Thus far, we have used one-dimensional arrays to model collections of elements…

Higher-dimensional arrays are also supported by Java.

Two-dimensional arrays can be used to represent matrices or tables. For example, the following table which describes the distances between 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

Objectives

- To give examples of representing data using two-dimensional arrays (§8.1).
- To declare variables for two-dimensional arrays, create arrays, and access array elements in a two-dimensional array using row and column indexes (§8.2).
- To program common operations for two-dimensional arrays (§8.3).
- To pass two-dimensional arrays to methods (§8.4).
- To return two-dimensional arrays from methods (§8.4).
- Higher dimensional arrays

Declare/Create Two-dimensional Arrays

Syntax:

Declare and create a two-dimensional array:

```
dataType[][] refVar = new dataType[nrows][ncols];
```

Example:

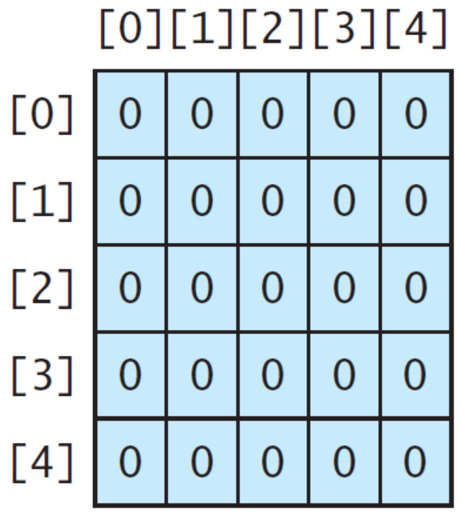
A two-dimensional array with 10 rows and 15 columns:

```
int[][] matrix = new int[10][15];
```

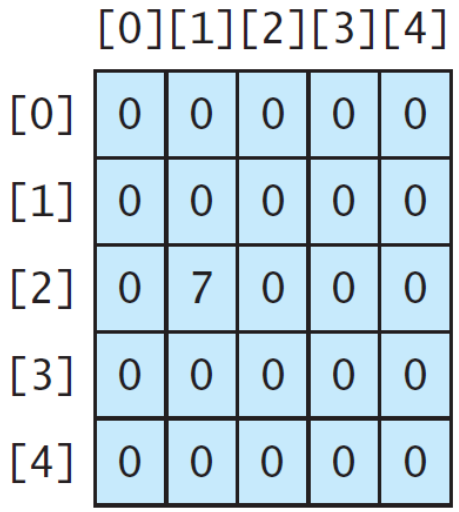
Two-dimensional Array Example

Example: declare and create a two-dimensional array with five rows and five columns and assign a value to one array element:

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



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



Declaring, Creating, and Initializing Arrays Using Shorthand Notation

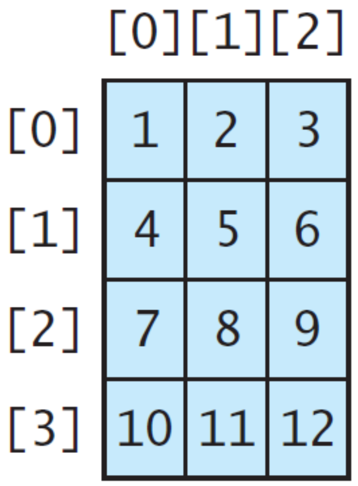
	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9
[3]	10	11	12

Declaring, Creating, and Initializing Arrays Using Shorthand Notation

$$\begin{array}{c|c|c|c|}
 & [0] & [1] & [2] \\
\hline
[0] & 1 & 2 & 3 \\
\hline
[1] & 4 & 5 & 6 \\
\hline
[2] & 7 & 8 & 9 \\
\hline
[3] & 10 & 11 & 12 \\
\hline
\end{array}$$

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

Declaring, Creating, and Initializing Arrays Using Shorthand Notation

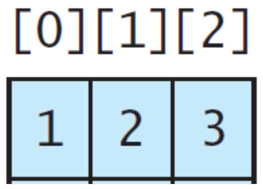


```
int[][] a = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

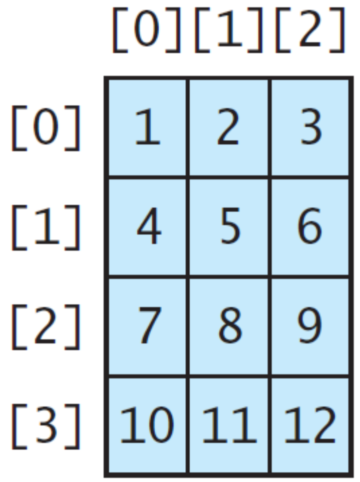
Same as

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

One- versus Two-dimensional Arrays



```
int[] arr = {1, 2, 3};
```



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

Lengths of Two-dimensional Arrays

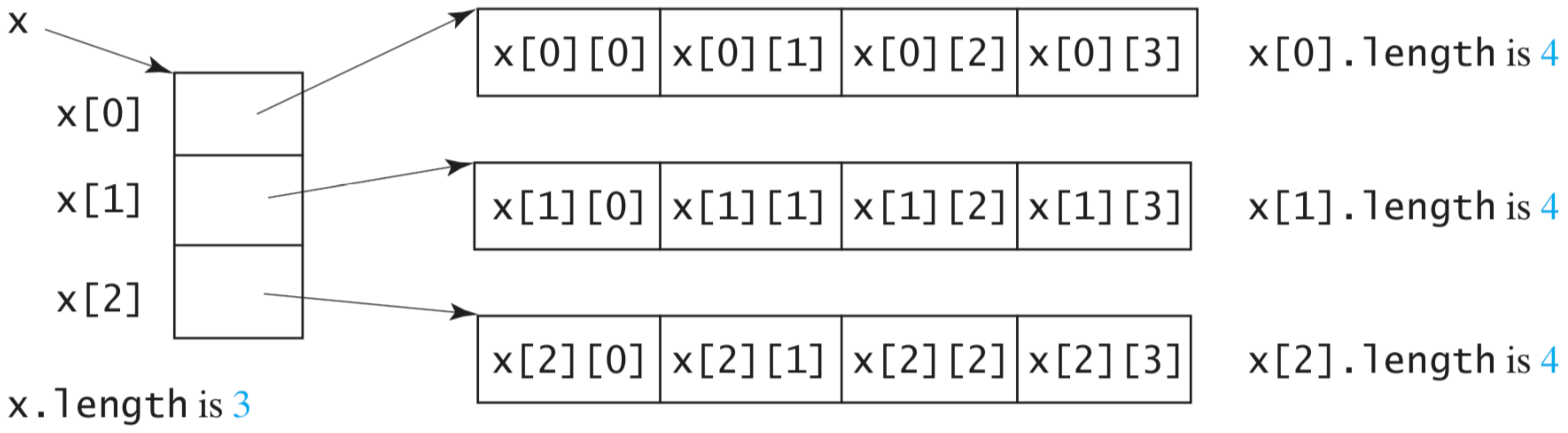
A two-dimensional array is actually a one-dimensional array, elements of which contain references to other one-dimensional arrays

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

Lengths of Two-dimensional Arrays

A two-dimensional array is actually a one-dimensional array, elements of which contain references to other one-dimensional arrays

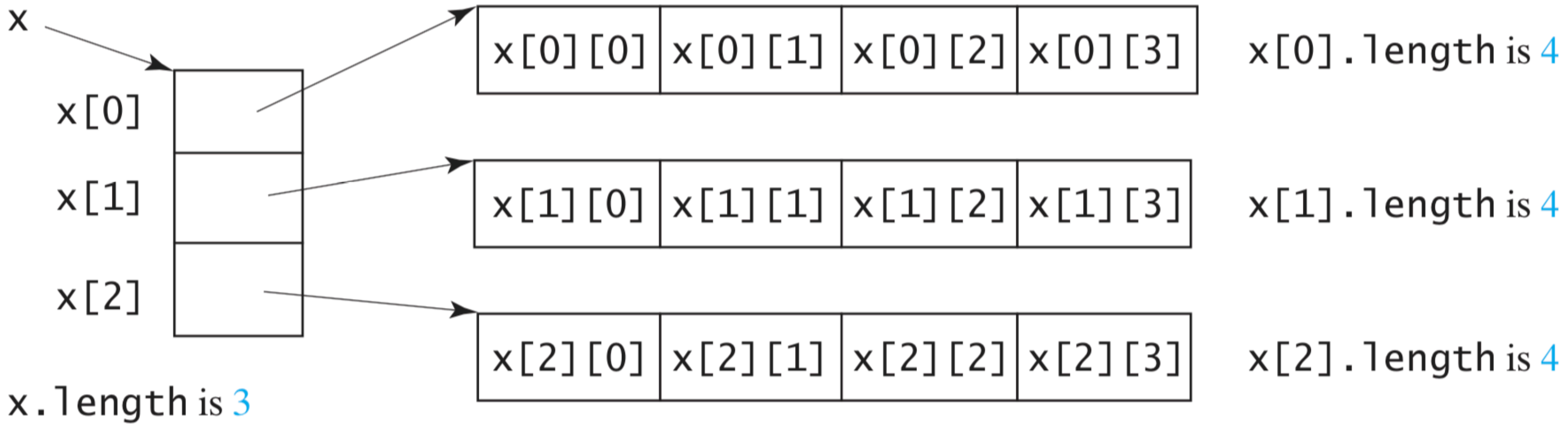
```
int[][] x = new int[3][4];
```



Lengths of Two-dimensional Arrays

A two-dimensional array is actually a one-dimensional array, elements of which contain references to other one-dimensional arrays

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



x
x[0]
x[1]
x[2]
x[0][0] x[0][1] x[0][2] x[0][3]
x[1][0] x[1][1] x[1][2] x[1][3]
x[2][0] x[2][1] x[2][2] x[2][3]
x[0].length is 4
x[1].length is 4
x[2].length is 4
x.length is 3

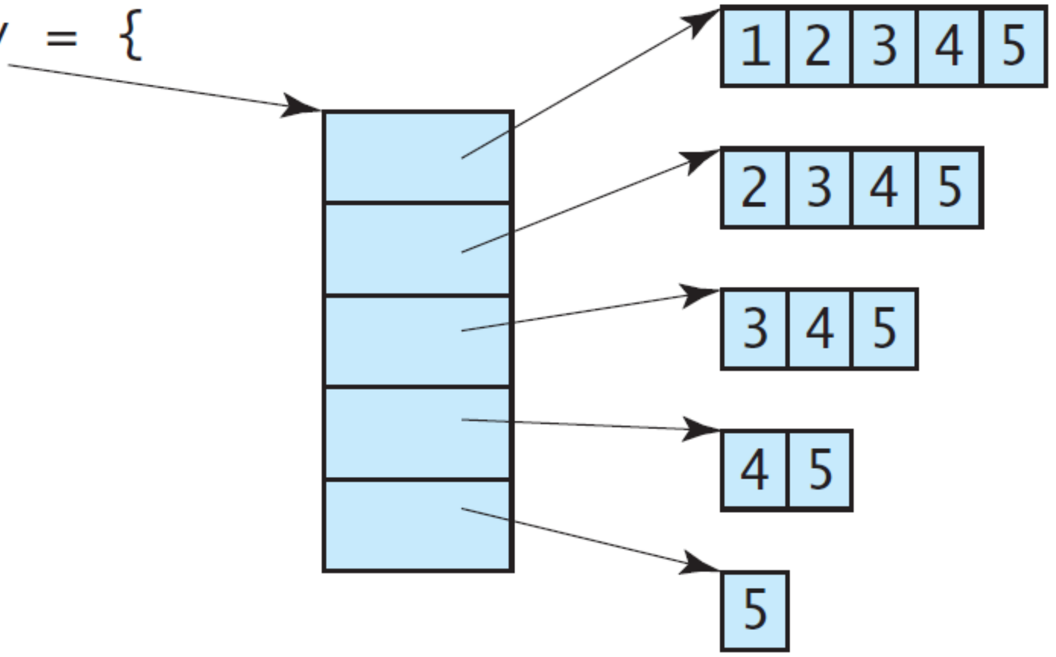
```
x[3].length => ArrayIndexOutOfBoundsException
```

Ragged Arrays

Each row in a two-dimensional array is itself an array… So, the rows can have different lengths.

Such an array is known as *a ragged array*. For example:

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



Ragged Arrays

If the values in a ragged array are not known in advance, but you do know the size of each row, a ragged array can be created as follows:

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

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

Processing Two-Dimensional Arrays

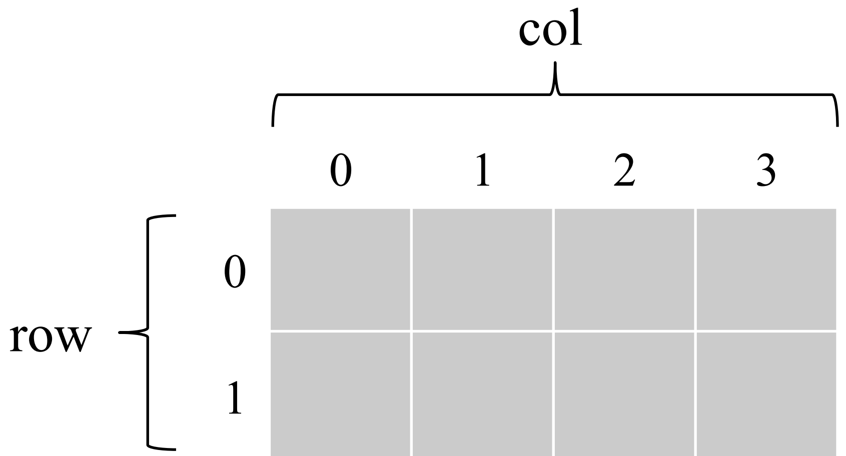
Some examples:

1. Initializing arrays with random values
2. Printing arrays
3. Summing all elements
4. Check if an array is symmetric

Initializing arrays with random values

```
int[][] matrix = new int[2][4];
...

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

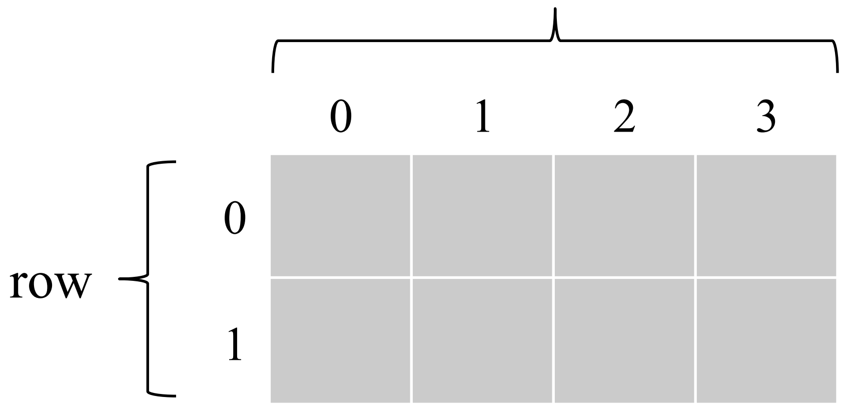


Printing arrays

```
int[][] matrix = new int[2][4];
...

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

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

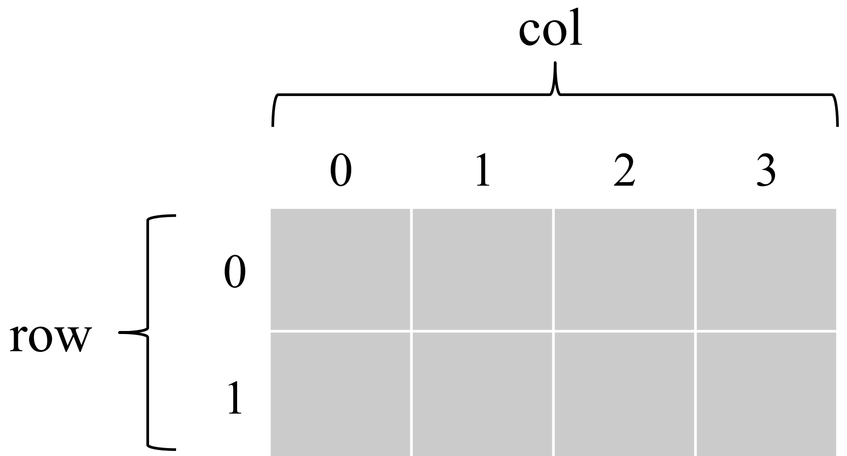


Summing all elements

```
int[][] matrix = new int[2][4];
...

int sum = 0;

for (int row = 0; row < matrix.length; row++)
  for (int col = 0; col < matrix[row].length; col++)
    sum += matrix[row][col];
```



Check if an array is symmetric

```
int[][] matrix = new int[nrows][ncols];
...

// Check if the matrix is square
boolean isSymmetric = nrows == ncols;

// Compare each element above and below the main diagonal
if(isSymmetric) {
  for (int row = 1; row < matrix.length; row++) {
    for (int col = 0; col < row; col++)
      if (matrix[row][col] != matrix[col][row]) {
        isSymmetric = false;
        break;
      }

    if(!isSymmetric)
      break;
  }
}

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

	0	**1**	**2**	**3**
0	5	1	3	7
1	1	2	8	5
2	3	8	7	4
3	7	5	4	3

Two-Dimensional Arrays Example

Example program showing how to:

- Pass a two-dimensional array to a method
- Return a two-dimensional array from a method

PassTwoDimensionalArray

Multidimensional Arrays

Previously – seen examples of using two-dimensional arrays to represent tables and matrices…

In Java, you can create *n*-dimensional arrays for any integer *n* >= 2:

- A two-dimensional array consists of an array of one-dimensional arrays
- A three-dimensional array consists of an array of two-dimensional arrays
- …

For example, use a three-dimensional array to store exam scores for a class of **six** students with **five** exams, and each exam has **two** parts (multiple-choice and essay):

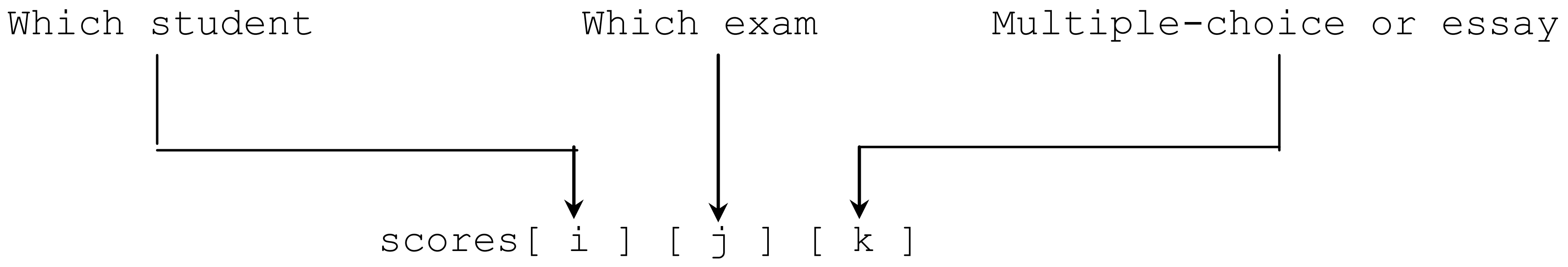
```
double[][][] scores = new double[6][5][2];
```

Multidimensional Arrays

```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.2, 21.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}
};
```

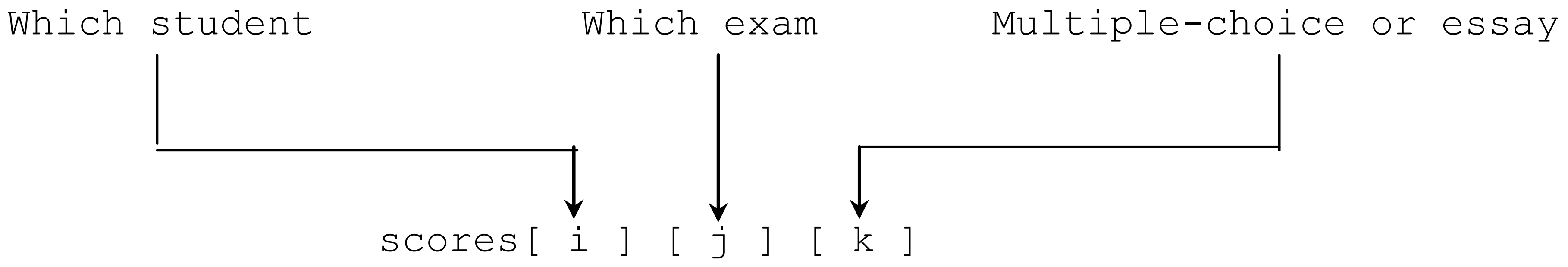
Multidimensional Arrays

```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.2, 21.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}
};
```



Multidimensional Arrays

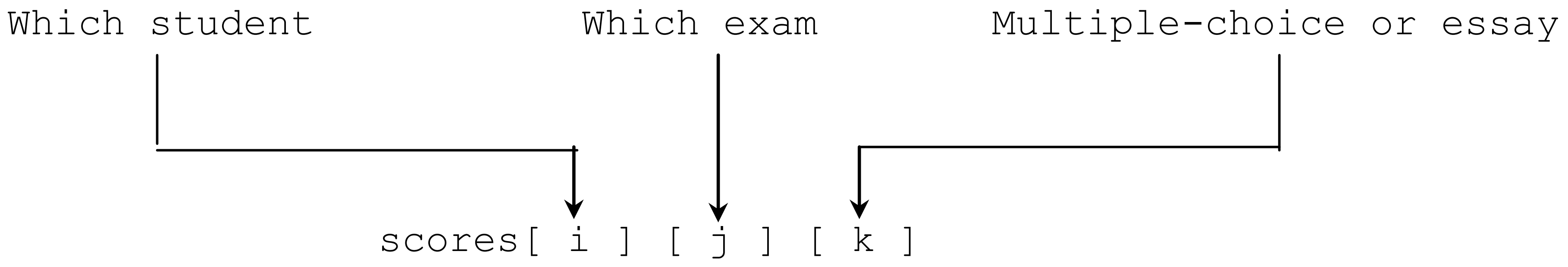
```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.2, 21.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}
};
```



`scores[0][1][0]` refers to ??

Multidimensional Arrays

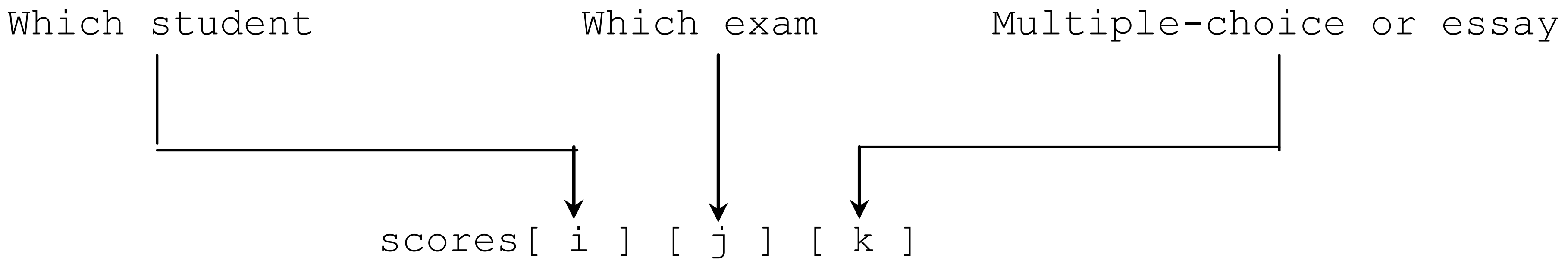
```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.2, 21.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}
};
```



`scores[0][1][0]` refers to the multiple-choice score for the first student's second exam, which is **9.0**.

Multidimensional Arrays

```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.2, 21.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}
};
```

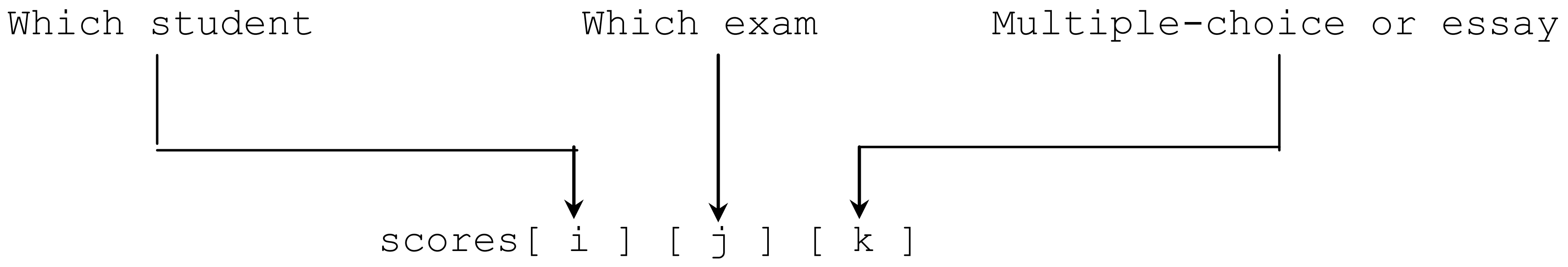


`scores[0][1][0]` refers to the multiple-choice score for the first student's second exam, which is 9.0.

`scores[0][1][1]` refers to ??

Multidimensional Arrays

```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.2, 21.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}
};
```



`scores[0][1][0]` refers to the multiple-choice score for the first student's second exam, which is 9.0.

`scores[0][1][1]` refers to the essay score for the first student's second exam, which is **22.5**.

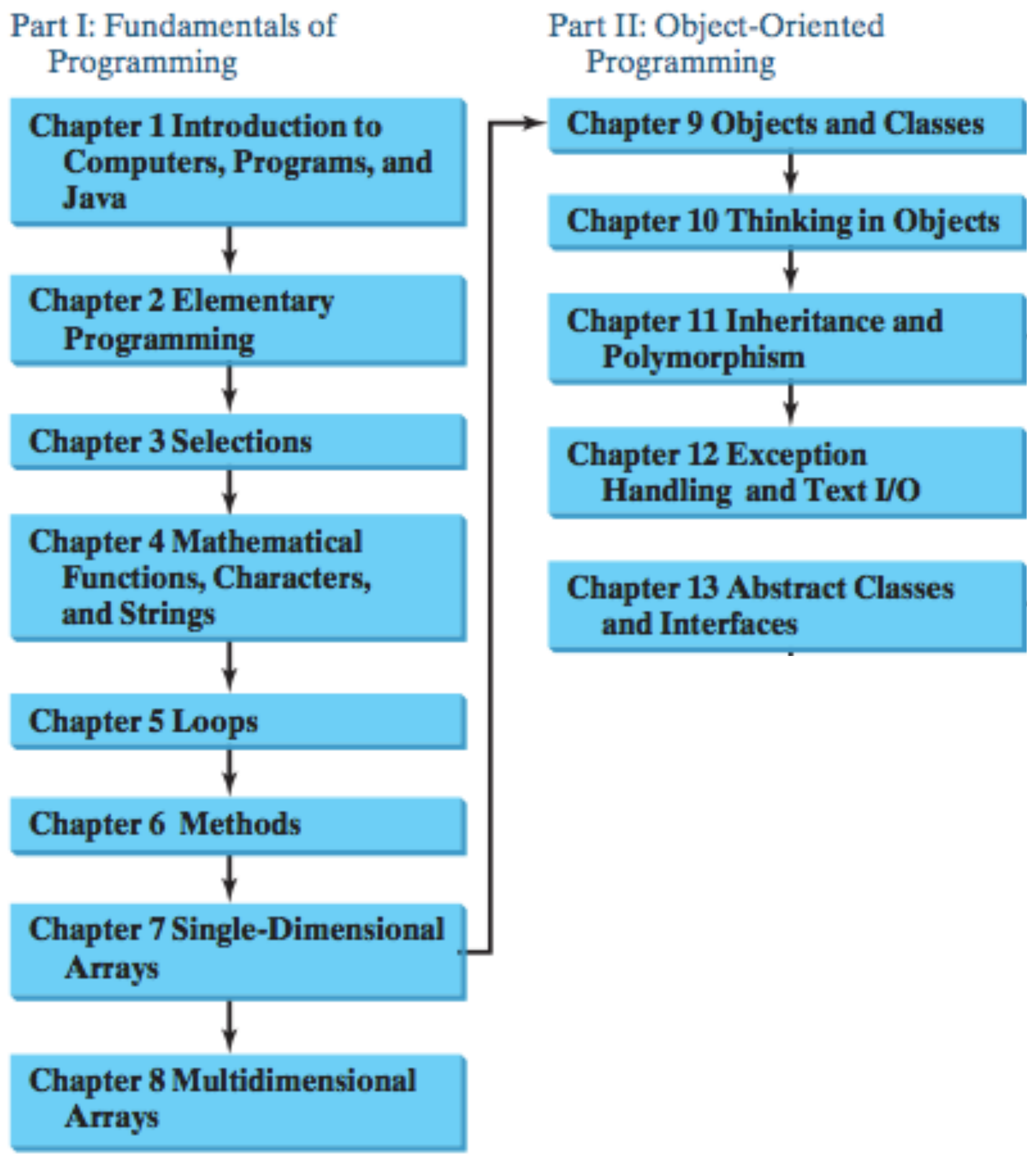
Next topics

Part II: Object-orientated Programming

Chapter 9:

- Objects and Classes

Topics covered so far...


Part I: Fundamentals of Programming
Chapter 1 Introduction to Computers, Programs, and Java
Chapter 2 Elementary Programming
Chapter 3 Selections
Chapter 4 Mathematical Functions, Characters, and Strings
Chapter 5 Loops
Chapter 6 Methods
Chapter 7 Single-Dimensional Arrays
Chapter 8 Multidimensional Arrays
Part II: Object-Oriented Programming
Chapter 9 Objects and Classes
Chapter 10 Thinking in Objects
Chapter 11 Inheritance and Polymorphism
Chapter 12 Exception Handling and Text I/O
Chapter 13 Abstract Classes and Interfaces