Advanced Programming

Arrays

ThS. Trần Thị Thanh Nga
Khoa CNTT, Trường ĐH Nông Lâm TPHCM
Email: ngattt@hcmuaf.edu.vn

Array Basics

- An array is used to store a **collection of data**
- An array as a **collection of variables of the same type**.
- Instead of declaring individual variables, such as **number0**, **number1**, and **number99**,
  - you declare one array variable such as numbers and use **numbers[0]**, **numbers[1]**, and **numbers[99]** to represent individual variables.

Declaring Array Variables

- Syntax:

  elementType[] arrayRefVar;

- Example:

  double[] myList;

Creating Arrays

- The declaration of an array variable does not allocate any space in memory for the array.
  - It creates only a storage location for the **reference** to an array.
- If a variable does not contain a reference to an array, the value of the variable is **null**.
- You cannot assign elements to an array unless it has already been created.

Creating Arrays

- Creating an array by using the **new** operator.
- Syntax:

  arrayRefVar = **new** elementType[arraySize];

- This statement does 2 things:
  - (1) it creates an array using **new elementType[array-Size];**
  - (2) it assigns the **reference** of the newly created array to the variable **arrayRefVar**.

Creating Arrays

- **Declaring** an array variable, **creating** an array, and **assigning** the reference of the array to the variable can be combined:

  elementType[] arrayRefVar = **new** elementType[arraySize];

or: **elementType** arrayRefVar[] = **new** elementType[arraySize];

- Example:

  double[] myList = **new double**[10];

- To assign values to the elements, use the syntax:

  arrayRefVar[index] = value;

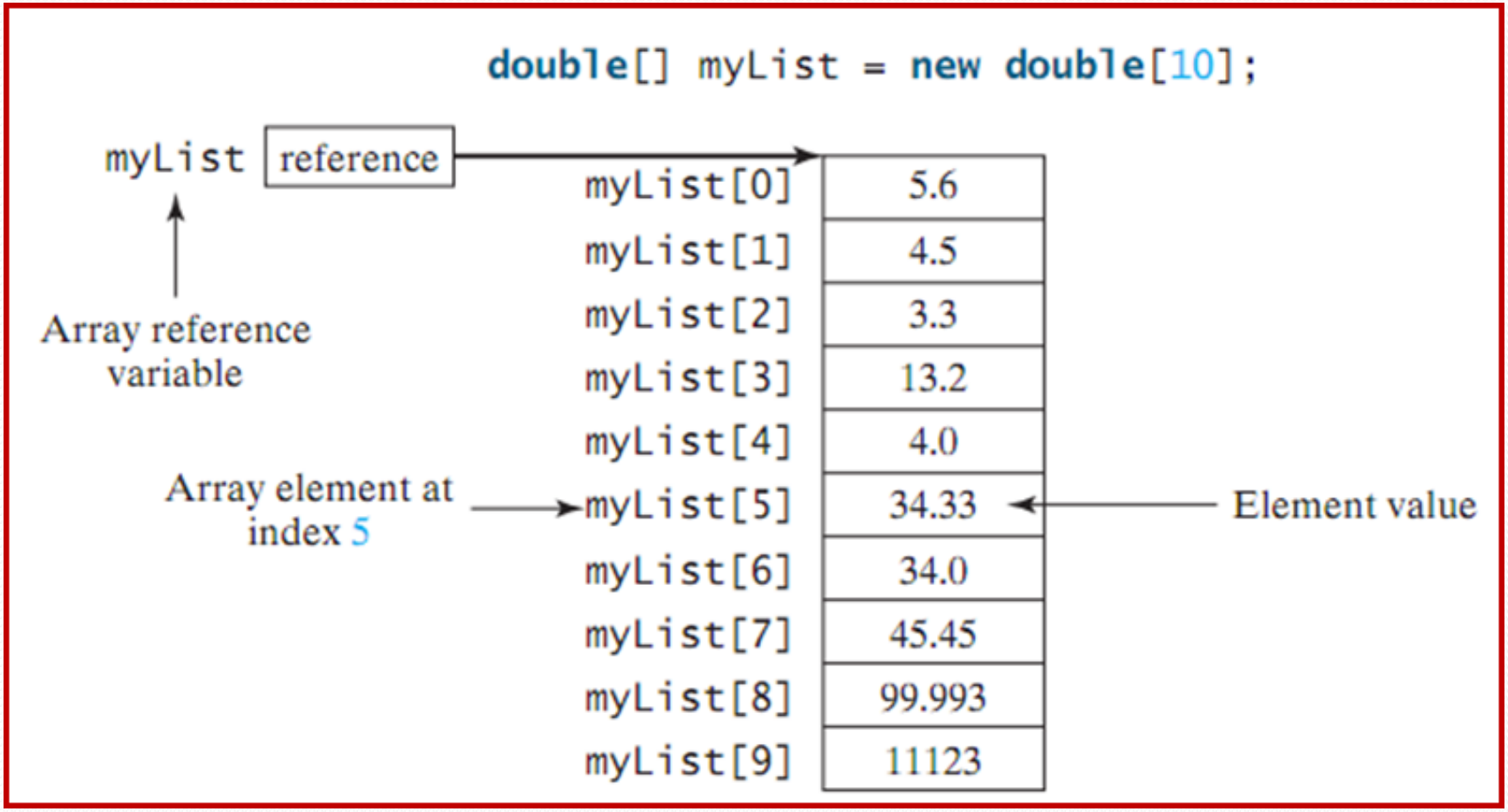
Assigning Arrays

- To assign values to the elements, use the syntax:

  arrayRefVar[index] = value;

```
myList[0] = 5.6;
myList[1] = 4.5;
myList[2] = 3.3;
myList[3] = 13.2;
myList[4] = 4.0;
myList[5] = 34.33;
myList[6] = 34.0;
myList[7] = 45.45;
myList[8] = 99.993;
myList[9] = 11123;
```

Example of Array



Array Size and Default Values

- When space for an array is allocated, the array size must be given, specifying the number of elements that can be stored in it.
- The size of an array **cannot** be changed after the array is created.
- Size can be obtained using **myList.length**.

Array Indexed Variables

- The array elements are accessed through the index.
- They range from **0** to **arrayRefVar.length-1**.
- Each element in the array is represented using the following syntax, known as an indexed variable:

```
arrayRefVar[index];
```

- An indexed variable can be used in the same way as a regular variable:

```
myList[2] = myList[0] + myList[1];
```

Array Initializers

- Java has a shorthand notation, known as the **array initializer**, which combines in one statement **declaring** an array, **creating** an array, and **initializing**.

elementType[] arrayRefVar = {value**0**, value**1**, ..., value**k**};

- Example:
  - **double**[] myList = {**1.9**, **2.9**, **3.4**, **3.5**};
  - Equivalent to the statements shown below:

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

Processing Arrays

- When processing array elements, you will often use a **for** loop—for two reasons:
  - All of the elements in an array are of the *same type*. They are evenly processed in the same fashion repeatedly using a loop.
  - The **size** of the array is known, it is natural to use a **for** loop.

Example: Initializing arrays with input values

```
double[] myList = new double[10];
Scanner input = new Scanner(System.in);
System.out.print("Enter " + myList.length +“
                 values: ");
for (int i = 0; i < myList.length; i++)
     myList[i] = input.nextDouble();
```

Example: Initializing arrays with random values

```
double[] myList = new double[10];
for (int i = 0; i < myList.length; i++) {
     myList[i] = Math.random() * 100;
}
```

Example: Displaying arrays

```
for (int i = 0; i < myList.length; i++) {
      System.out.print(myList[i] + "\t");
}
```

Example: Summing all elements

```
double[] myList = new double[10];
//gán giá trị cho từng phần tử mảng
//…
double total = 0;
for (int i = 0; i < myList.length; i++) {
     total += myList[i];
}
```

Example: Finding the largest element

```
double[] myList = new double[10];
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
     if (myList[i] > max)
          max = myList[i];
}
```

Example: Finding the smallest index of the largest element

```
double[] myList = new double[10];
double max = myList[0];
int indexOfMax = 0;
for (int i = 1; i < myList.length; i++) {
     if (myList[i] > max) {
          max = myList[i];
          indexOfMax = i;
     }
}
```

Example: Random shuffling

```
double[] myList = new double[10];
for (int i = 0; i < myList.length; i++) {
      // Generate an index j randomly
      int index = (int) (Math.random() *
                         myList.length);
      // Swap myList[i] with myList[j]
      double temp = myList[i];
      myList[i] = myList[index];
      myList[index] = temp;
}
```

Example: Shifting elements

```
double[] myList = new double[10];
double temp = myList[0]; //Retain the first element
// Shift elements left
for (int i = 1; i < myList.length; i++) {
     myList[i - 1] = myList[i];
}
// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```

For-each Loops

- You can traverse the array sequentially without using an index variable.

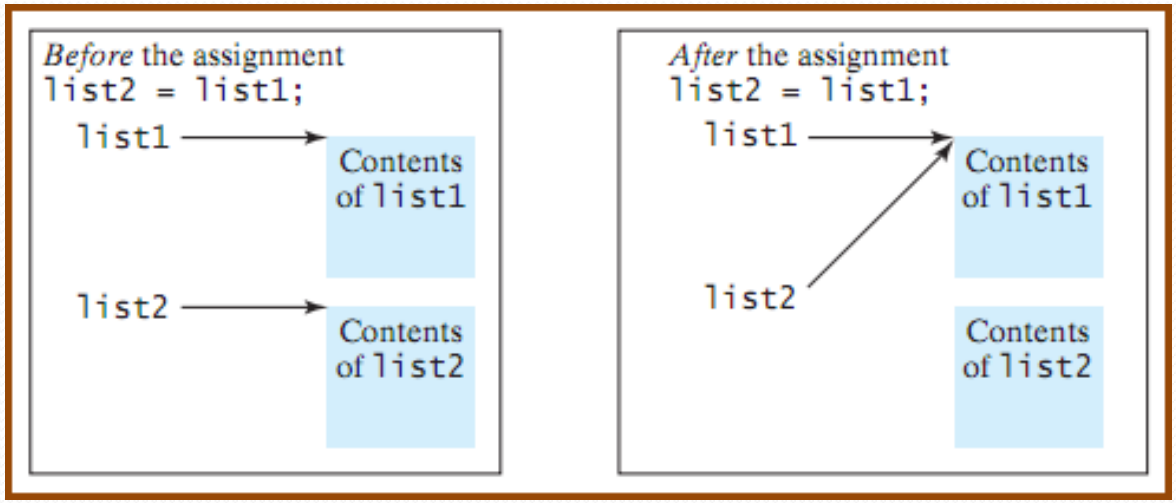
```
for (double u: myList) {
    System.out.println(u);
}
```

- "for each element u in myList do the following."
- the variable, **u**, must be declared the same type as the elements in **myList**

Copying Arrays

- You need to duplicate an array or a part of an array.
- You could attempt to use the assignment statement (=), as follows:

list2 = list1;



Copying Arrays

- Assigning one array variable to another array variable actually copies one reference to another and makes both variables point to the same memory location.
- There are three ways to copy arrays:
  - Use a loop to copy individual elements one by one.
  - Use the static **arraycopy** method in the **System** class.
  - Use the **clone** method to copy arrays.

Copying Arrays

```
int[] list1 = {1, 2};
int[] list2 = list1.clone();
list1[0] = 7; list1[1] = 8;
System.out.println("list1 is " + list1[0] + ",
                   " + list1[1]);
System.out.println("list2 is " + list2[0] + ",
                   " + list2[1]);
```

Passing Arrays to Methods

```
public static void printArray(int[] array) {
     for (int i = 0; i < array.length; i++)
     {
          System.out.print(array[i] + " ");
     }
}
```

You can invoke it by passing an array:

printArray(new int[]{3, 1, 2, 6, 4, 2});

Pass-by-value

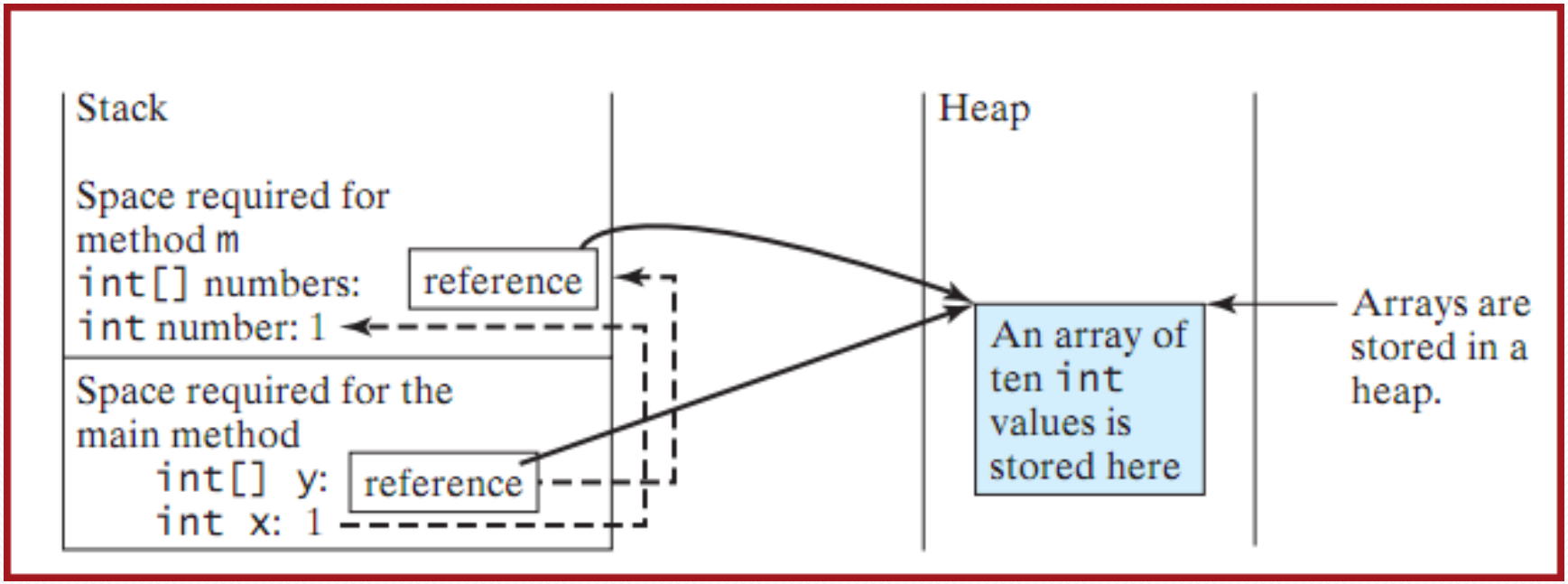
- Java uses **pass-by-value** to pass arguments to a method.
- There are important differences between passing the values of variables of **primitive** data types and passing **arrays**.
  - For an argument of a **primitive** type, the argument's value is passed.
  - For an argument of an **array** type: the value of the argument is a **reference** to an array; this reference value is passed to the method.
    - Semantically, it can be best described as pass-by-sharing.

Pass-by-value

```
public class Test {
   public static void main(String[] args) {
        int x = 1; // x represents an int value
        int[] y = new int[10]; // y represents an array of int values
        methodE(x, y); // Invoke m with arguments x and y
        System.out.println("x is " + x);
        System.out.println("y[0] is " + y[0]);
   }

   public static void methodE(int number, int[] numbers) {
        number = 1001; // Assign a new value to number
        numbers[0] = 5555; // Assign a new value to numbers[0]
   }
}
```

Pass-by-value



*The primitive type value in **x** is passed to **number**, and the reference value in **y** is passed to **numbers**.*

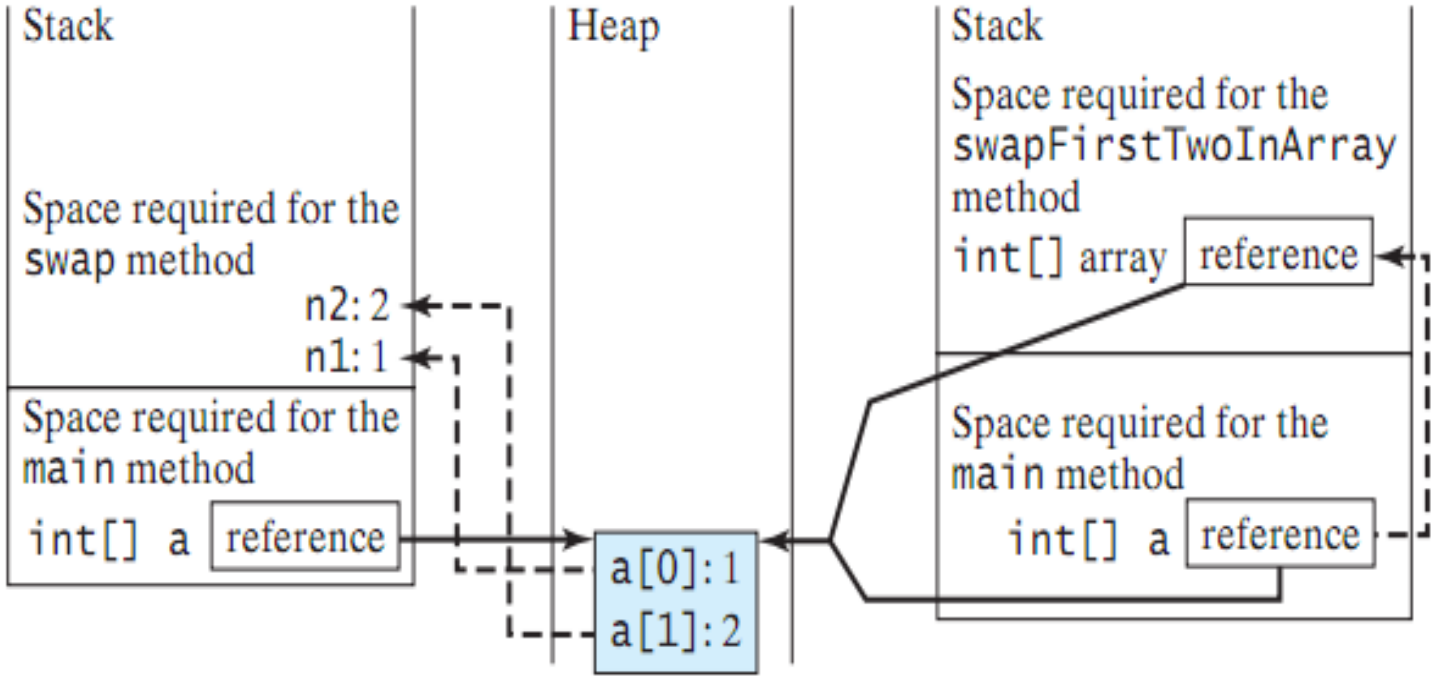
```
public class TestPassArray {
   public static void main(String[] args) {
      int[] a = { 1, 2 };
      // Swap elements using the swap method
      System.out.println("Before invoking swap");
      System.out.println("array is {" + a[0] + ", " + a[1] + "}");
      swap(a[0], a[1]);

      System.out.println("After invoking swap");
      System.out.println("array is {" + a[0] + ", " + a[1] + "}");
      // Swap elements using the swapFirstTwoInArray method
      System.out.println("Before invoking swapFirstTwoInArray");
      System.out.println("array is {" + a[0] + ", " + a[1] + "}");

      System.out.println("After invoking swapFirstTwoInArray");
      System.out.println("array is {" + a[0] + ", " + a[1] + "}");
   }
```

```
    /** Swap two variables */
    public static void swap(int n1, int n2) {
        int temp = n1;
        n1 = n2;
        n2 = temp;
    }

    /** Swap the first two elements in the array */
    public static void swapFirstTwoInArray(int[] array) {
        int temp = array[0];
        array[0] = array[1];
        array[1] = temp;
    }
}
```



Invoke `swap(int n1, int n2)`. The primitive type values in `a[0]` and `a[1]` are passed to the `swap` method.

The arrays are stored in a heap.

Invoke `swapFirstTwoInArray(int[] array)`. The reference value in `a` is passed to the `swapFirstTwoInArray` method.

Example

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

Declaring and Creating Two-Dimensional Arrays

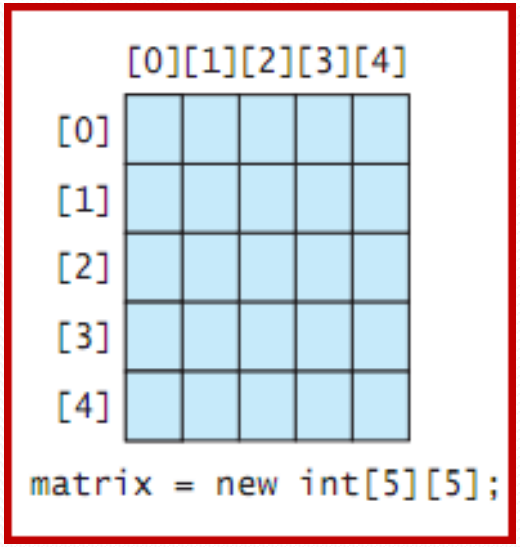
- Declaring a two-dimensional array

  elementType[][] arrayRefVar;

- Example: **int**[][] matrix;

- Create a two-dimensional array of 5-by-5 int values and assign it to matrix

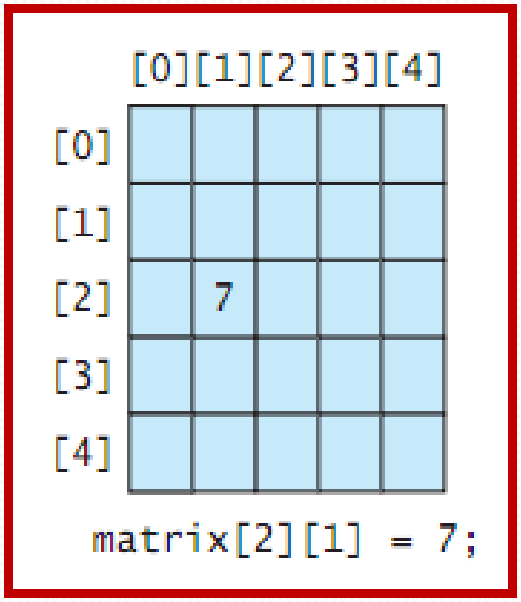
  matrix = new **int**[5][5];



Declaring and Creating Two-Dimensional Arrays

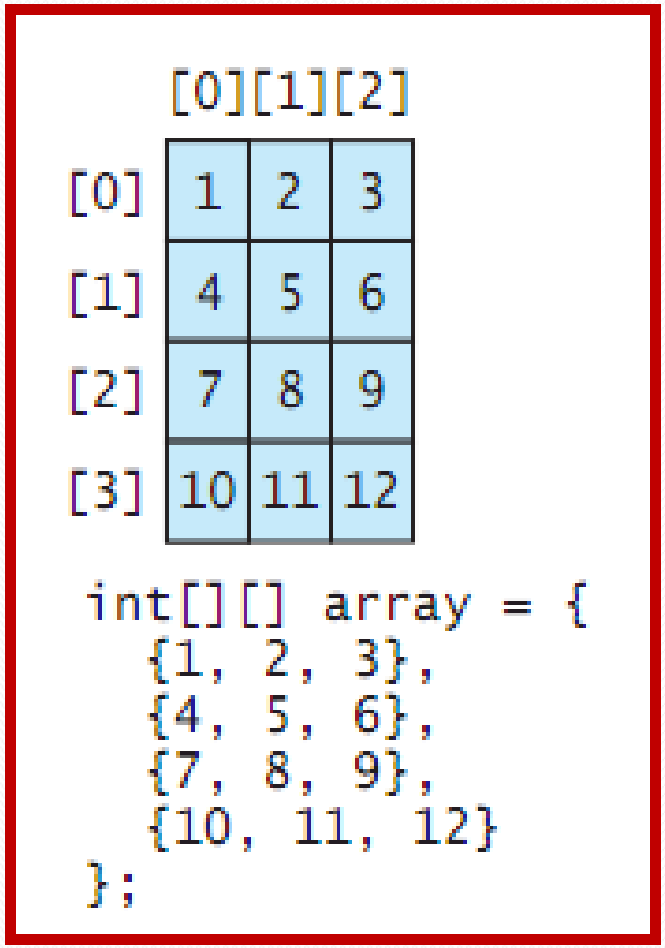
- To assign the value 7 to a specific element at row 2 and column 1:

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



Declaring and Creating Two-Dimensional Arrays

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



Declaring and Creating Two-Dimensional Arrays

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

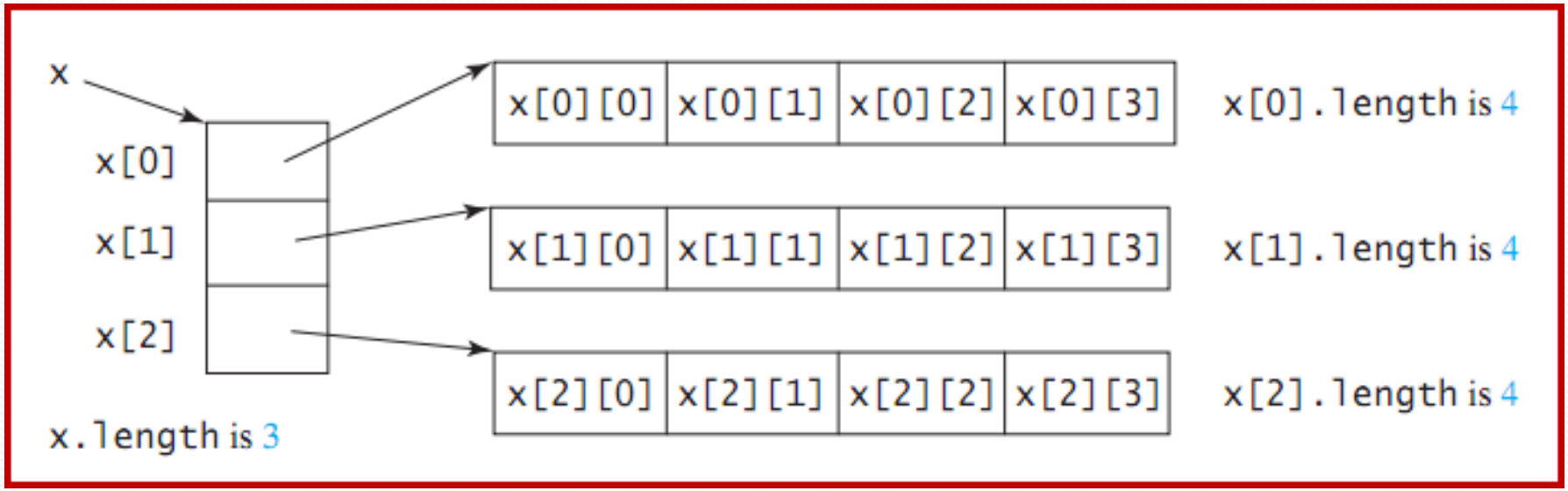
Equivalent

```
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;
```

Obtaining the Lengths of Two-Dimensional Arrays

- A **two-dimensional** array is actually an array in which each element is a **one-dimensional** array.
- The **length** of an array **x** is the number of elements in the array, which can be obtained using **x.length**.
- **x[0]**, **x[1]**, and **x[x.length-1]** are arrays.
  - Their lengths: x[0].length, x[1].length, and x[x.length-1].length

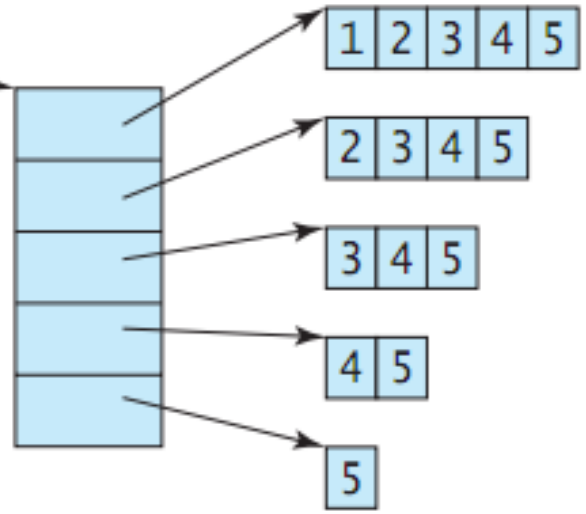
Obtaining the Lengths of Two-Dimensional Arrays



Ragged Arrays

- Each row in a two-dimensional array is itself an array.
- The rows can have different lengths ➔ **ragged array**.

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



Ragged Arrays

- Create a ragged array if you don't know the **values** in a ragged array, but know the **sizes**:

```
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];
```

- Assign values to the array

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

Initializing arrays with input values

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

```
int[][] matrix = new int[10][10];
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] = (int)(Math.random() * 100);
        }
}
```

Printing arrays

```
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

```
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 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);
}
```

Which row has the largest sum?

```
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);
```

Exercises: Array

- **Bài 1**: Cho n số nguyên
  - Tìm vị trí và giá trị phần tử lớn nhất của dãy.
  - Tìm vị trí và giá trị phần tử nhỏ nhất của dãy.
  - Tính tổng các phần tử của dãy.
- **Bài 2**: Cho n số nguyên. Tìm xem phần tử lớn nhất xuất hiện trong dãy mấy lần.
- **Bài 3**: Nhập vào n số nguyên
  - Đếm số phần âm, dương, bằng 0 của dãy.
  - Xác định số âm lớn nhất và số dương nhỏ nhất.
  - Cho biết | tổng âm| có bằng tổng dương không.

Exercises: Array (2)

- **Bài 4**: Cho n số. Đảo thứ tự của dãy theo nguyên tắc sau: A[0] đổi cho với A[n-1], A[1] đổi cho với A[n-2], ... In kết quả ra màn hình.
- **Bài 5**: Cho n số và số x.
  - Xác định xem số x có xuất hiện trong dãy không?
  - Cho biết số x xuất hiện trong dãy bao nhiêu lần và tại các vị trí nào?
  - Loại bỏ khỏi dãy tất cả các phần tử bằng x. In cả 2 dãy ra màn hình.

Exercises: Array (3)

- **Bài 6**: Cho một dãy gồm n số.
  - Kiểm tra xem dãy có tăng dần hay không.
  - Sắp xếp dãy theo thứ tự tăng dần. In dãy kết quả ra màn hình.
- **Bài 7**: Cho một dãy gồm n số.
  - Kiểm tra xem dãy có đối xứng hay không. Ví dụ dãy sau là đối xứng: 4 2 7 3 7 2 4
  - Kiểm tra xem dãy có đan dấu hay không. Ví dụ dãy sau là đan dấu: 2 -1 7 -3 4 -5 6

Exercises: Array (4)

- **Bài 8**: Cho ma trận các số nguyên kích thước m x n:
  - In ra phần tử lớn nhất và nhỏ nhất của ma trận.
  - Tính tổng các phần tử của ma trận.
- **Bài 9**: Cho ma trận vuông A cấp n gồm các số nguyên. Tính tổng từng dòng của ma trận và tìm dòng có tổng lớn nhất.

Exercises: Array (5)

- **Bài 10**: Cho ma trận vuông cấp n gồm các số nguyên.
  - Kiểm tra ma trận có là ma trận tam giác trên không? (Ma trận tam giác trên thỏa: ít nhất một phần tử phía trên đường chéo chính khác 0, và toàn bộ các phần tử dưới đường chéo chính bằng 0).
  - Kiểm tra ma trận có đối xứng qua đường chéo chính hay không.
  - Kiểm tra ma trận có đối xứng qua tâm hay không.

Reference

- **Introduction to Java Programming** 8th , Y. Daniel Liang.