UNSW Business School
Information Systems and Technology Management

INFS1609/2609 Fundamentals of Business Programming

Lecture 7
Arrays

yenni.tim@unsw.edu.au

INFS1609/2609 Fundamentals of Business Programming

Topics for this week:

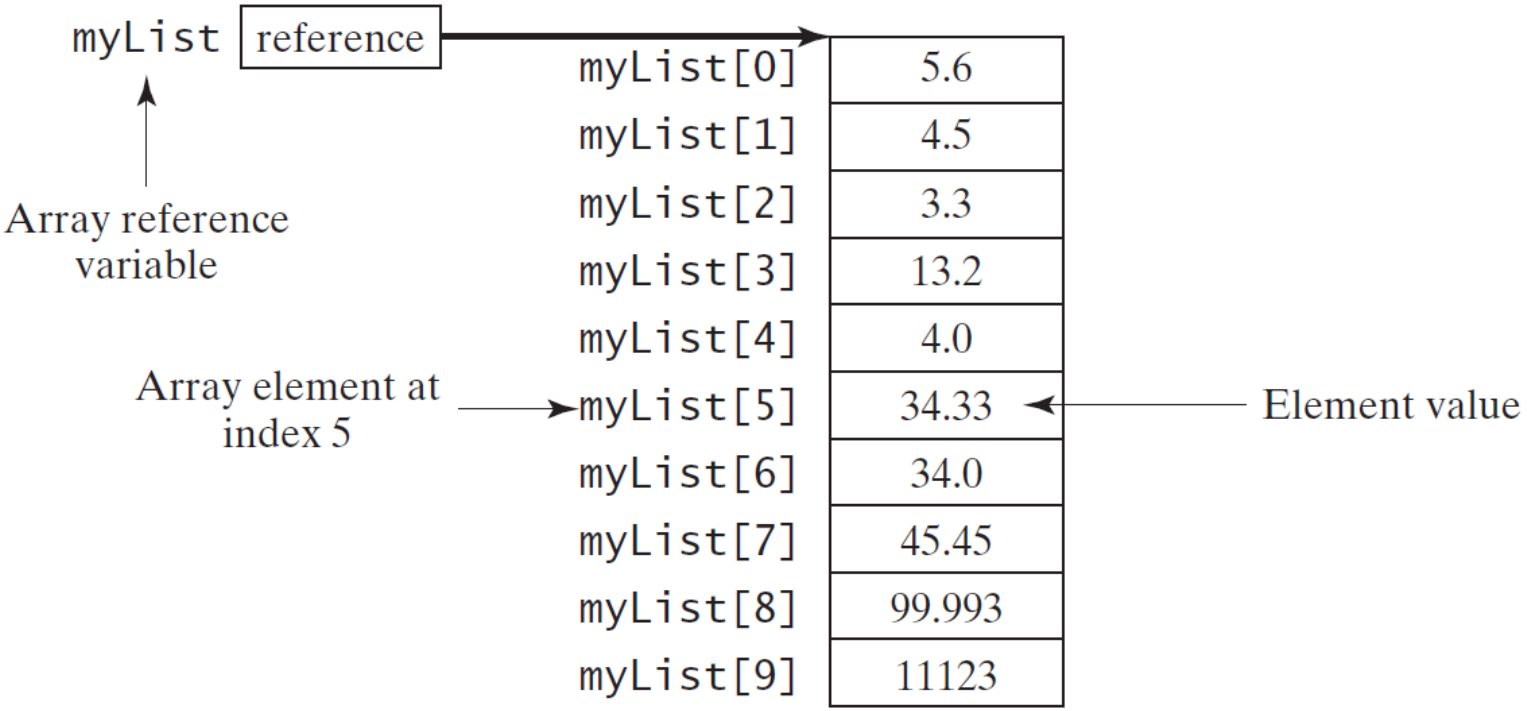
- To understand why arrays are necessary in programming
- To declare array reference variables and create arrays
- To access array elements using indexes obtain array size using .length
- To program common array operations (displaying arrays, summing all elements, finding the minimum and maximum elements, random shuffling, and shifting elements)
- To develop and invoke methods with array arguments and return values

Main references
- *Textbook: Chapter 7*
- *Other online references posted on Ed*

Introducing Arrays

Array is a data structure that represents a collection of the same types of data

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



INFS1609/2609 Fundamentals of Business Programming

Introducing Arrays

Syntax:

```
datatype[] arrayName = new datatype[arraySize];
```

Example:

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

This statement **declares** an array variable, `myList`, creates **an array** of ten elements of `double` type, and assigns its **reference** to `myList`

INFS1609/2609 Fundamentals of Business Programming

Introducing Arrays

Example

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

`myList[0]` references the **first** element in the array
`myList[9]` references the **last** element in the array

Introducing Arrays

Declaring, creating and initializing arrays in one step

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This is equivalent to the following statements:

```
double[] myList = new double[4];

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 3.4;

myList[3] = 3.5;
```

Introducing Arrays

Using the shorthand notation, you have to declare, create, and initialize the array all in **one statement**. Splitting it would cause a syntax error. For example, the following is wrong:

```
double[] myList;

myList = {1.9, 2.9, 3.4, 3.5};
```

Introducing Arrays

Once an array is created, its size is fixed and cannot be changed. You can find the size of an array using:

```
arrayName.length
```

For example,

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

`myList.length` returns 10

Introducing Arrays

When an array is created, its elements are assigned the default value of

0 for the numeric primitive data types,
'\u0000' (null character) for char types, and
false for boolean types

Indexed Variables

The array elements are accessed through the index. The array indices are *0-based*, i.e., **it starts from 0** to **array.length-1**

In the example, `myList` holds ten double values and the indices are from 0 to 9

Each element in the array is represented using the following syntax, known as an *indexed variable*:

```
arrayName[index];
```

Indexed Variables

After an array is created, an indexed variable can be used in the same way as a regular variable. For example, the following code adds the value in `myList[0]` and `myList[1]` to `myList[2]`

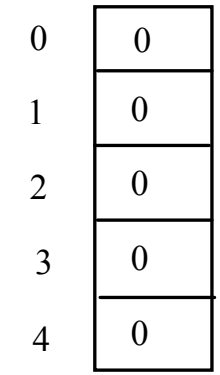
```
myList[2] = myList[0] + myList[1];
```

Tracing Program with arrays

Declare array variable values, create an array, and assign its **reference** to values

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created



Tracing Program with arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

0	11
1	1
2	3
3	6
4	10

The final array

Examples: Working with arrays using for loop

Going through every elements in the array

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

Examples: Working with arrays using for loop

Finding the largest element in an array

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

Examples: Working with arrays using for loop

Initializing arrays with input values

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

java.util.Scanner input = new java.util.Scanner(System.in);

System.out.println("Enter " + myList.length + " values: ");

for(int i = 0; i < myList.length; i++){
    myList[i] = input.nextInt();
}
```

Examples: Working with arrays using for loop

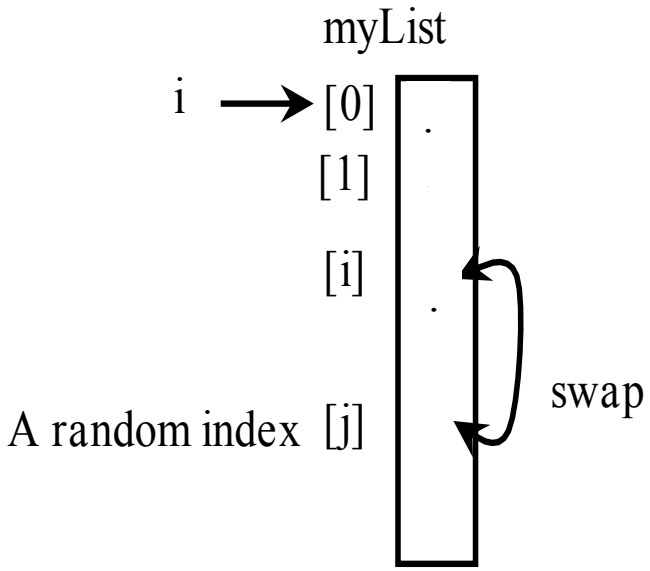
Summing all elements in an array

```
double total = 0;
for (int i = 0; i < myList.length; i++) {
   total += myList[i];
}
```

Examples: Working with arrays using for loop

Random shuffling

```
for (int i = 0; i < myList.length; i++) {
  // Generate an index j randomly
  int j = (int)(Math.random()
    * myList.length);

  // Swap myList[i] with myList[j]
  double temp = myList[i];
  myList[i] = myList[j];
  myList[j] = temp;
}
```



A different way to iterate through items in an array

Enhanced for Loop (for-each loop)

- JDK 1.5 introduced a new for loop that enables you to traverse the complete array sequentially without using an index variable

```
for (elementType value: arrayRefVar) {
  // Process the value
}
```

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

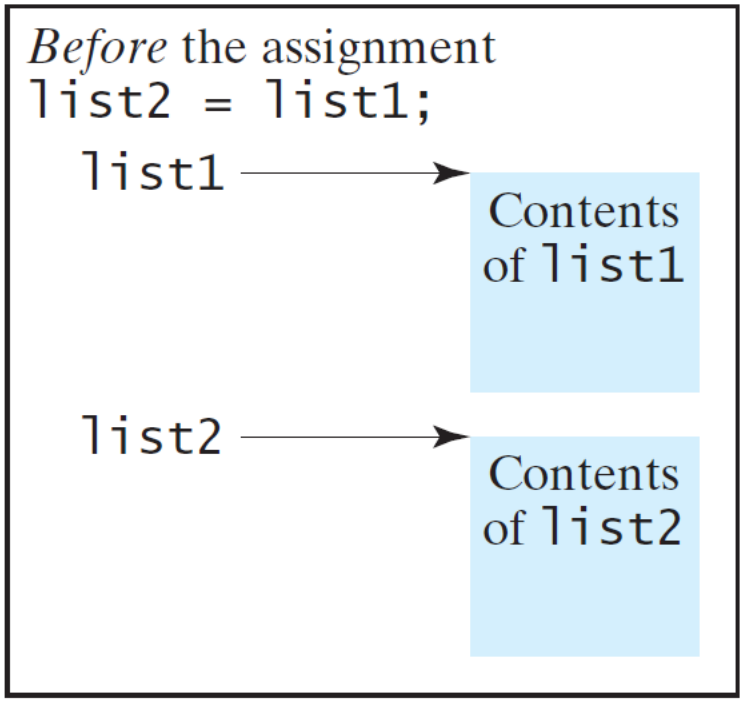
Let's look at more examples!

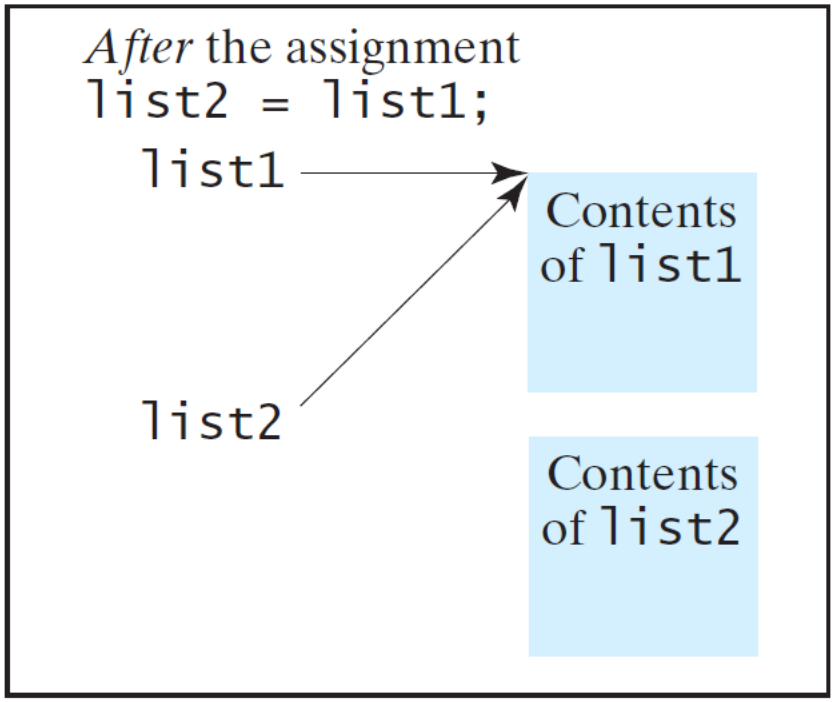
Read up to one hundred numbers, compute their average, and find out how many numbers are above the average

https://liveexample.pearsoncmg.com/html/AnalyzeNumbers.html

Copying Arrays

The assignment operator (=)





Copying Arrays

Using a loop

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new
int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++){
   targetArray[i] = sourceArray[i];
}
```

Copying Arrays

The `arraycopy` utility

```
arraycopy(sourceArray, src_pos, targetArray,
tar_pos, length);
```

Example:

```
System.arraycopy(sourceArray, 0, targetArray,
0, sourceArray.length);
```

INFS1609/2609 Fundamentals of Business Programming

Copying Arrays

How are the three copy methods different?

Passing Arrays to Methods

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

```
Invoke the method

int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

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

Anonymous array

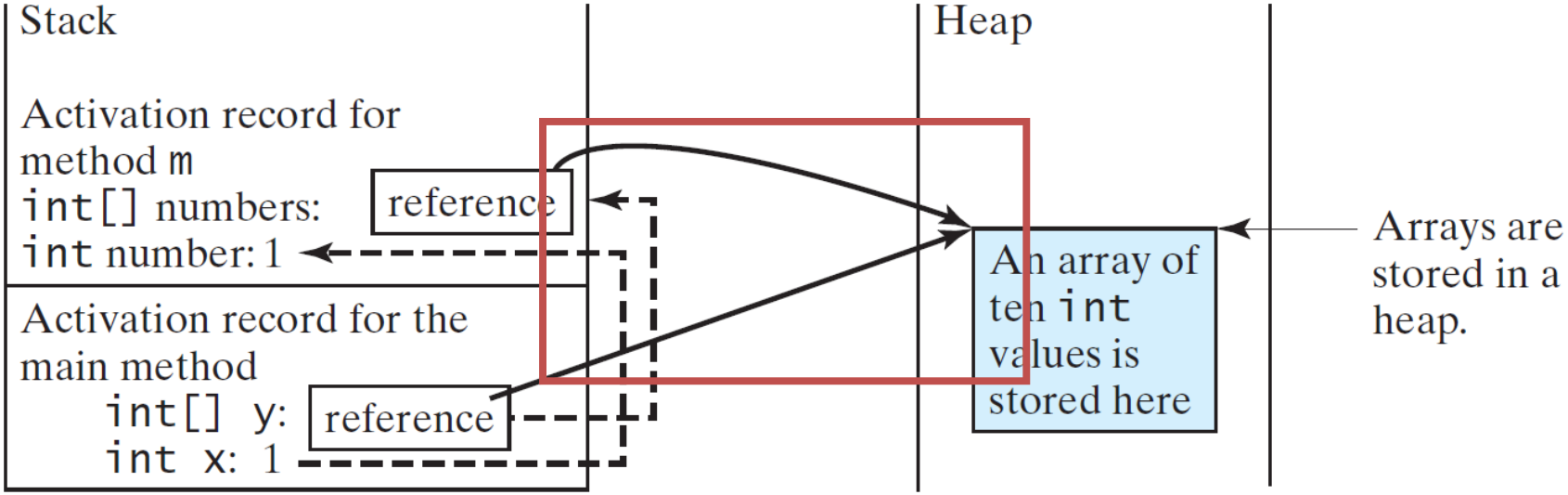
Pass by value

- There are important differences between passing a value of variables of primitive data types and passing arrays (reference type)
- For a parameter of a primitive type value, the **actual value** is passed. Changing the value of the local parameter inside the method **does not** affect the value of the variable outside the method
- For a parameter of an array type, **the value of the parameter contains a reference** to an array; this reference is passed to the method. Any changes to the array that occur inside the method body **will affect** the original array that was passed as the argument

Simple Example

```
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

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

Simple Example



When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, **numbers now contains the same reference value to the same array**.

INFS1609/2609 Fundamentals of Business Programming

Demonstrate differences of passing **primitive data type** variables and **array (reference)** variables

Code example and supplementary videos on Ed

Returning an array from a method

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

INFS1609/2609 Fundamentals of Business Programming

Combining the many things you have learned so far…

Write a program that generates 10 lowercase letters randomly and assign to an array of characters.

Count the occurrence of each letter in the array

Multi-dimensional Arrays

- You have used one-dimensional arrays to model linear collections of elements
- You can use a two-dimensional array to represent a matrix or a table

Distance Table (in miles)

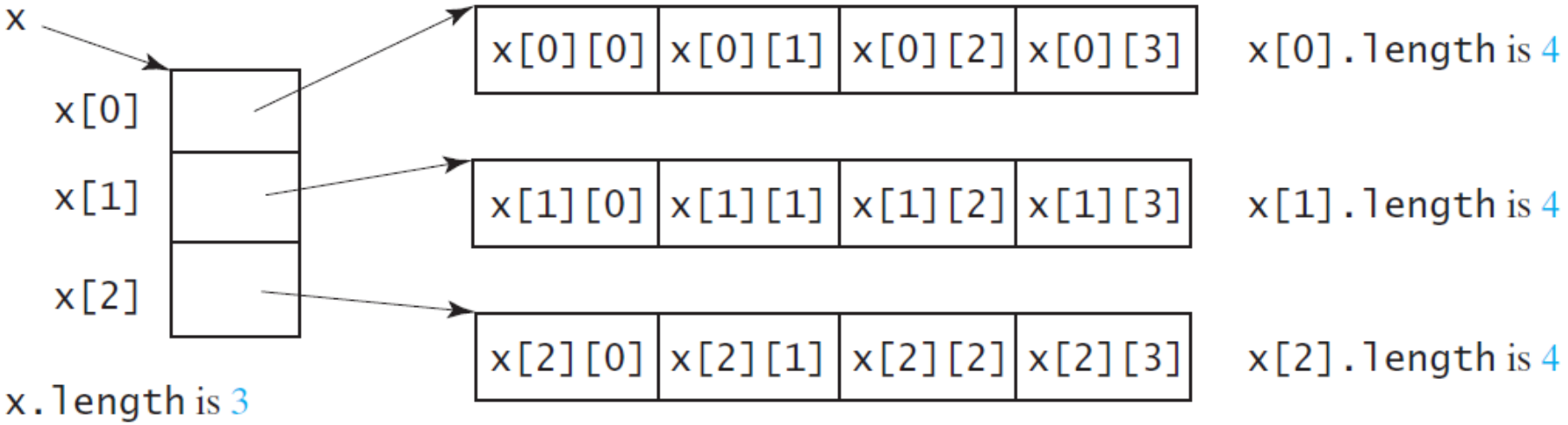
	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187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

Declaring two-dimensional Arrays

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

// Create array and assign its reference to variable
refVar = new dataType[10][10];

// Combine declaration and creation in one statement
dataType[][] refVar = new dataType[10][10];

// Alternative syntax
dataType refVar[][] = new dataType[10][10];
```

Length of two-dimensional Arrays

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



Declaring two-dimensional Arrays

	[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)

array.length? 4

array[0].length? 3

Declaring two-dimensional Arrays

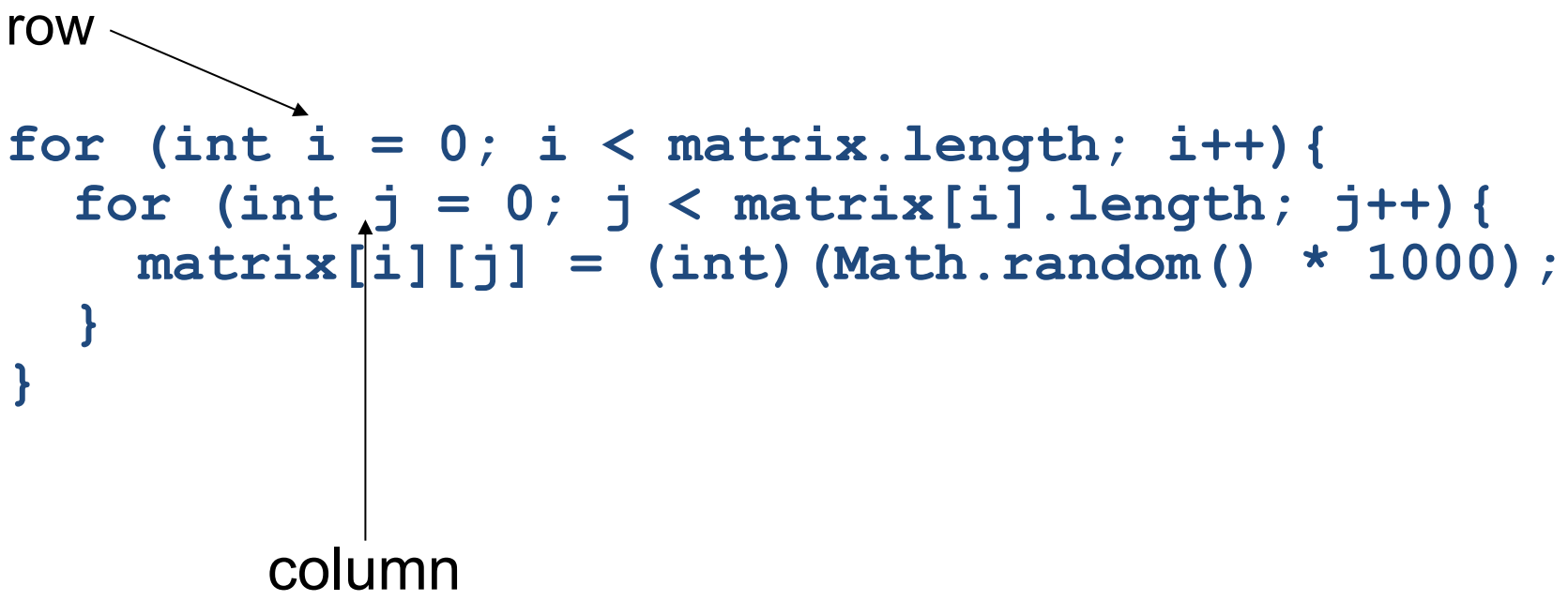
- You can also use an array initializer to declare, create and initialize a two-dimensional array. For example,

```
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;
```

Using for loops with two-dimensional Arrays



Printing two-dimensional 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();
}
```

Passing Two-dimensional Arrays to Methods

```
public static int sum(int[][] m) {
  int total = 0;
  for (int row = 0; row < m.length; row++) {
    for (int column = 0; column < m[row].length; column++) {
          total += m[row][column];
    }
  }
  return total;
}
```

https://liveexample.pearsoncmg.com/html/PassTwoDimensionalArray.html

INFS1609/2609 Fundamentals of Business Programming

If we have time…

Array versus **ArrayList**

- Array is a container object that holds a fixed number of values of a **single type**
- The length of an array is established when the array is created
- After creation, its **length is fixed**

Operation	*Array*	*ArrayList*
Creating an array/ArrayList	`String[] a = new String[10]`	`ArrayList<String> list = new ArrayList<>();`
Accessing an element	`a[index]`	`list.get(index);`
Updating an element	`a[index] = "London";`	`list.set(index, "London");`
Returning size	`a.length`	`list.size();`
Adding a new element		`list.add("London");`
Inserting a new element		`list.add(index, "London");`
Removing an element		`list.remove(index);`
Removing an element		`list.remove(Object);`
Removing all elements		`list.clear();`

The `ArrayList` Class

- A very useful class for storing objects
- ArrayList has the generic type `E`
- A concrete type can be specified to replace the **generic type**

Read:
https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

```
ArrayList<String> cities = new ArrayList<String>();
```

This ArrayList is used to store **strings**

java.util.ArrayList<E>	
+ArrayList()	Creates an empty list.
+add(o: E) : void	Appends a new element o at the end of this list.
+add(index: int, o: E) : void	Adds a new element o at the specified index in this list.
+clear(): void	Removes all the elements from this list.
+contains(o: Object): boolean	Returns true if this list contains the element o.
+get(index: int) : E	Returns the element from this list at the specified index.
+indexOf(o: Object) : int	Returns the index of the first matching element in this list.
+isEmpty(): boolean	Returns true if this list contains no elements.
+lastIndexOf(o: Object) : int	Returns the index of the last matching element in this list.
+remove(o: Object): boolean	Removes the element o from this list.
+size(): int	Returns the number of elements in this list.
+remove(index: int) : boolean	Removes the element at the specified index.
+set(index: int, o: E) : E	Sets the element at the specified index.

Learn how to work with Java docs:
https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

The `ArrayList` Class

- An ArrayList is just an object, and we can create it just like any other object:

```
ArrayList myList = new ArrayList();
//new ArrayList is initially empty
```

- To add an object to the ArrayList, we call the `add()` method provided by the ArrayList class

```
myList.add("New");
myList.add("Item");
```

The `ArrayList` Class

- ArrayList also uses zero-based indexing (take note of the use of `get` method to retrieve an element from the arraylist):

```
System.out.println(list.get(0)); //prints "New"
System.out.println(list.get(1)); //prints "Item"
```

- To get the size (in array, the length) of the arraylist, use the `size()` method

```
int listSize = myList.size();//size() returns 2
```

Reflection

What have you learned today?

Thank you!