OKANAGAN

COSC 111

Computer Programming I

Chapter 7 Single-Dimensional Arrays

Dr. Abdallah Mohamed

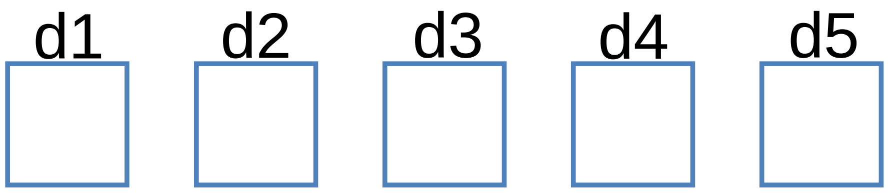
Acknowledgement: The slides mainly rely on the textbook of Y. D. Liang titled “Introduction to Java Programming, 10th Ed.”, Pearson Edu. Inc.

Arrays Overview

Suppose you need many variables in your program.

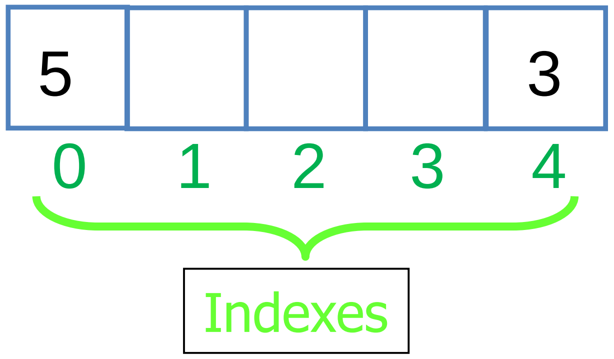
You could either create a separate name for each variable:

- `int d1, d2, d3, d4, d5;`



Or you could create an array that has multiple spots (indexes):

- `int[] d = new int[5];`
- `d[0] = 5;`
- `d[4]= 3;`

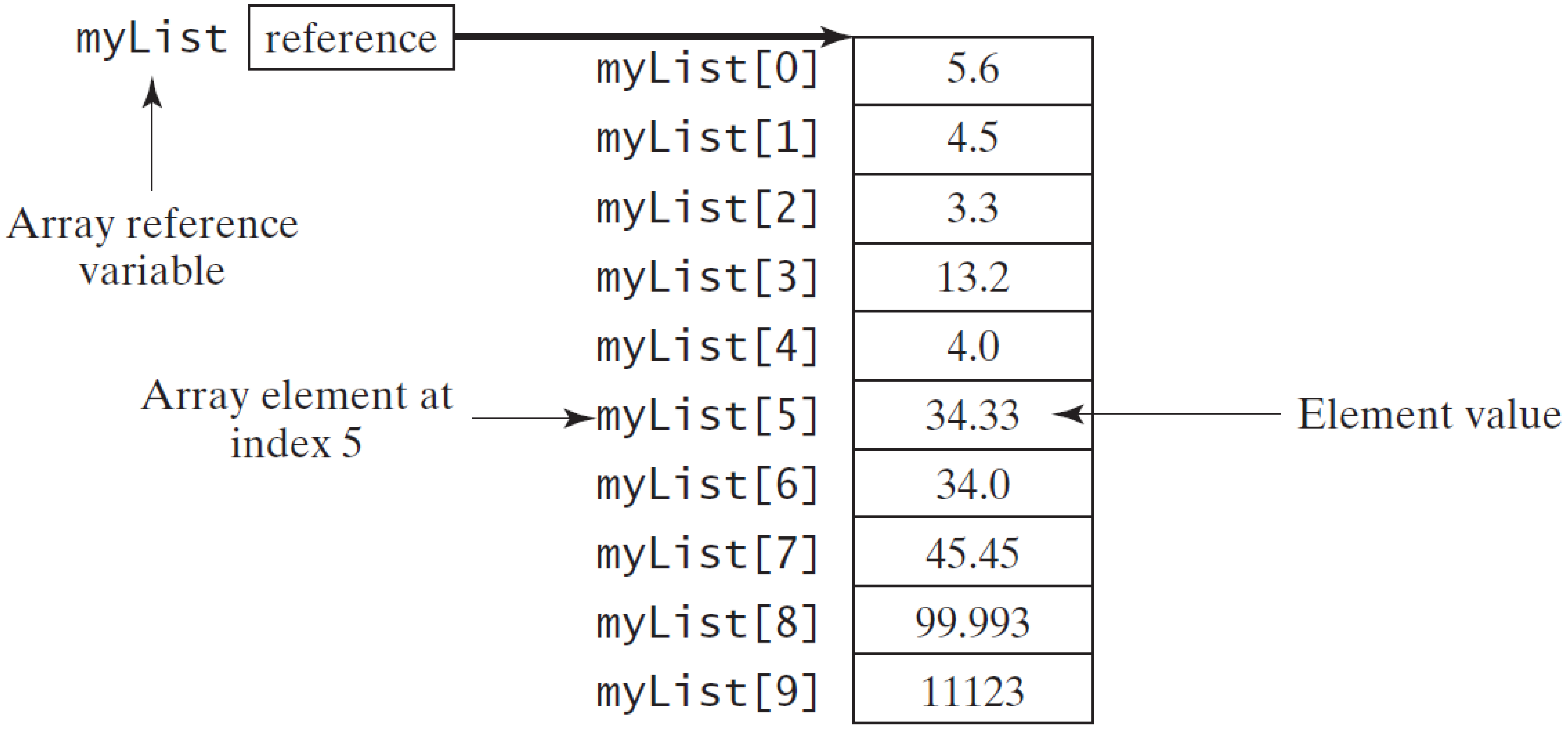


Introducing Arrays

Array is a **data structure** that represents a collection of the **SAME types of data**.

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

//code to initialize the array
```



Declaring, Creating, and Initializing Arrays

Creating an Array

(1) declare a **variable to *reference*** the array:

```
    int[] myList;
          //just the pointer – no array object yet.
```

(2) create the **array object**

```
    myList = new int[7];
          //Now myList is referring to a seven element array object
```

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

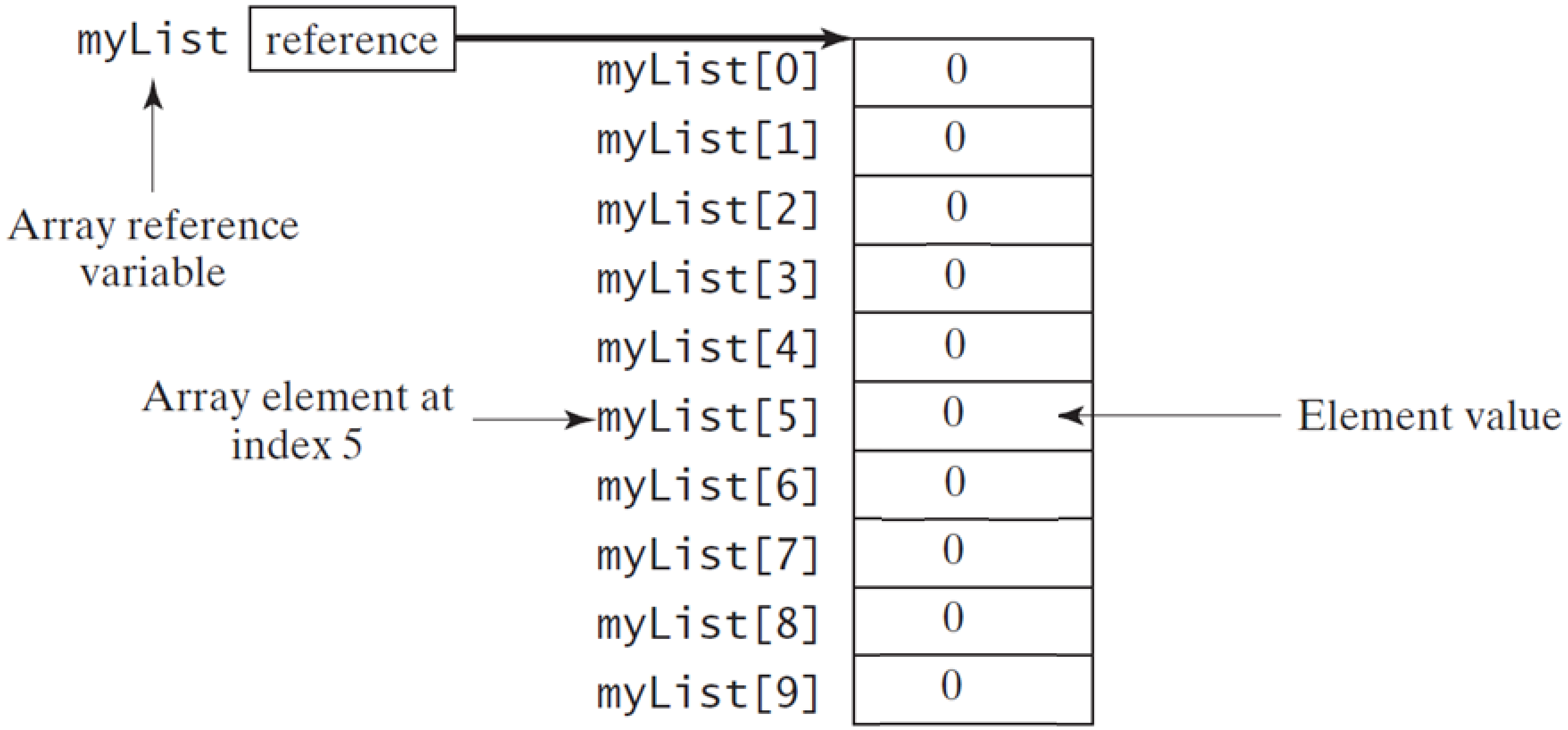
- **0** for the numeric primitive data types,
- **\u0000** for **char** types, and
- **false** for **boolean** types.

Creating Arrays, cont.

Example: `myList = new int[10];`

`myList[0]` references the first element in the array.

`myList[9]` references the last element in the array.



Creating Arrays, cont.

Declaring and creating arrays in one step:

Example: `int myList[] = new int[10];`

Declaring, creating, and *initializing* arrays in one step:

Example: `int myList[] = {5, 2, 10, 7, 12, 4, 12, 13, 8, 1};`

This shorthand notation is equivalent to the following statements:

```
int[] myList = new int[10];
myList[0] = 5;
myList[1] = 2;
…
myList[9] = 1;
```

Creating Arrays, cont.

CAUTION: Using the shorthand notation, you have to declare, create, and initialize the array **all in one statement**.

```
double[] myList;
myList = {2, 5, 3.4, 3.5}; //this is wrong!
```

The Length of an Array: Once an array is created, **its size is fixed**. You can find its size using, for example:

```
myList.length
```

Accessing Array Elements

The array indices are 0-based, i.e., it starts from 0 to *length-1*.

- myList, for example, has 10 int values, and the indices are from 0 to 9.

Example:

the following code adds the value in myList[0] and myList[1] to myList[2].

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

Practice

1) Create an `int` array with name `ages` with 20 elements.

2) Set the value of the 1st element to 12.

3) Set the value of the last element to 1.

4) Set the 2nd element to a value obtained from the user

5) How do you know how many elements are in an array?

Clicker Question

What is the size of this array?

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

A. 9

B. 10

C. 11

D. error

Clicker Question

What are the contents of this array?

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

A. error

B. 0, 1, 2, 3

C. 1, 2, 3, 4

D. 4, 3, 2, 1

Processing Arrays

Processing Arrays

1) Using a for loop: it is preferred to use a for loop to process array elements when several elements are processed in the **same fashion repeatedly**.

```
for (int i=0; i<arrayRefVar.length; i++){
    //same actions applied to all elements evenly
}
```

- **For example:** this code Initializes an array *ar* with random values :

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

2) Without a for loop: if you are applying **different actions** to the array elements, you should **probably avoid using the loop**.

Processing Arrays: Examples

Initializing an array with random values:

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

Initializing arrays with input values

```
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();
```

Printing arrays

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

Processing Arrays: Examples, cont.

Summing all elements

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

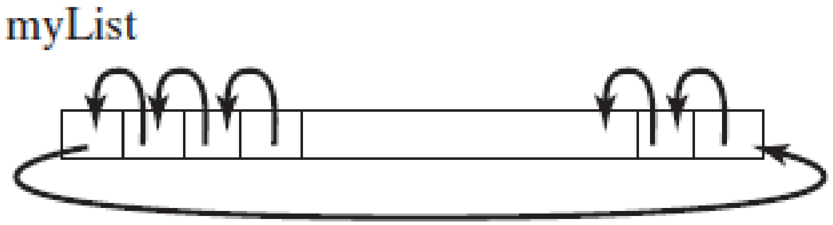
Finding the largest element

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

Processing Arrays, cont.

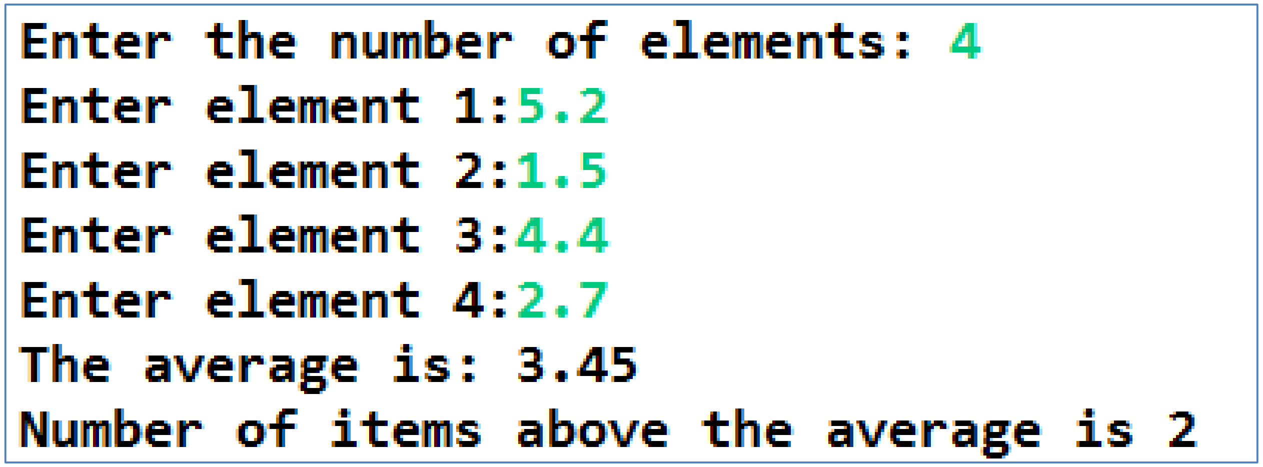
Shifting the elements one position to the left and filling the last element with the first element:

```
// Retain the first element
int temp = myList[0];
// 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;
```



Practice

Write a program that reads *n* elements (real numbers) and then displays their average **along with the number of elements above average**. Your program should begin by asking the use to enter the value *n.*



```
Enter the number of elements: 4
Enter element 1:5.2
Enter element 2:1.5
Enter element 3:4.4
Enter element 4:2.7
The average is: 3.45
Number of items above the average is 2
```

Practice, cont.

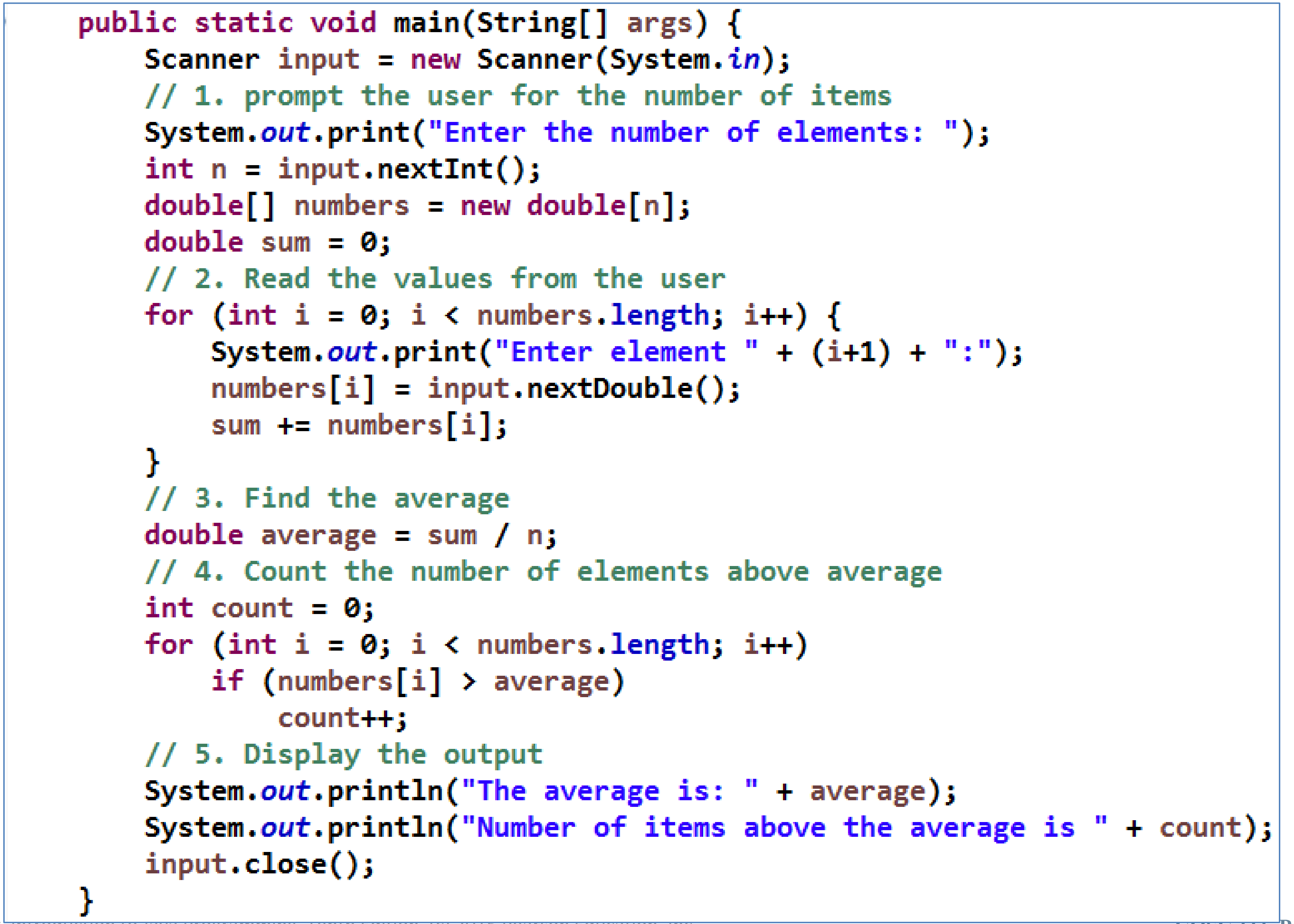
Basic idea:

If only the average is required, then we will run a simple for loop similar to what we did before, and we don’t need arrays. However, to get the number of items above the average, we need to remember these numbers at the end and compare them with the average. To remember the entries, store them in an array.

The algorithm:

- 1. Prompt the user for the number of items
- 2. Read the values from the user:
  - Store them in an array.
  - Find the sum
- 3. Find the average.
- 4. Count the number of elements above average.
- 5. Display the output.

Practice, cont.



```java
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    // 1. prompt the user for the number of items
    System.out.print("Enter the number of elements: ");
    int n = input.nextInt();
    double[] numbers = new double[n];
    double sum = 0;
    // 2. Read the values from the user
    for (int i = 0; i < numbers.length; i++) {
        System.out.print("Enter element " + (i+1) + ":");
        numbers[i] = input.nextDouble();
        sum += numbers[i];
    }
    // 3. Find the average
    double average = sum / n;
    // 4. Count the number of elements above average
    int count = 0;
    for (int i = 0; i < numbers.length; i++)
        if (numbers[i] > average)
            count++;
    // 5. Display the output
    System.out.println("The average is: " + average);
    System.out.println("Number of items above the average is " + count);
    input.close();
}
```

Clicker Question

What are the contents of this array?

```
int[] nums = new int[5];
for(int i = 0; i <= 5; i++)
    nums[i] = 5-i;
```

A. error

B. 0, 1, 2, 3, 4

C. 0, 1, 2, 3, 4, 5

D. 4, 3, 2, 1, 0

E. 5, 4, 3, 2, 1, 0

Clicker Question

What are the contents of this array?

```
int[] nums = new int[6];
for(int i = 2; i <= 4; i++)
    nums[i] = i-1;
```

A. error

B. 0, 1, 2, 3, 0, 0

C. 0, 0, 1, 2, 3, 0

D. 0, 0, 0, 1, 2, 3

E. 0, 1, 2, 3, 4, 5

Arrays are Objects!!

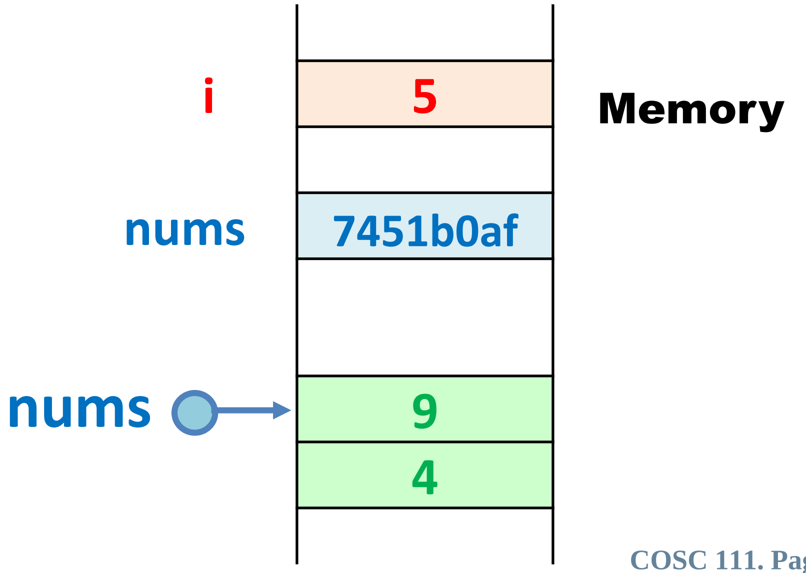
Arrays are Objects, not primitive types!

Java’s types are divided into:

- 1. Primitive types
  - Includes `boolean`,`byte`,`char`,`short`,`int`,`long`,`float` and `double`.
  - A primitive-type variable stores **a value** of its declared type.
- 2. Reference types
  - Includes all non-primitive types, (e.g., **Arrays**, Strings, Scanner, etc.)
  - A reference-type variable stores data which Java uses to find the object in the memory.
  - Such a variable is said to **refer to an object** in the program.

- Exampe:

```
int i = 5;
int[] nums = { 9 , 4 };
```



Declaring, Creating, and Working with Arrays

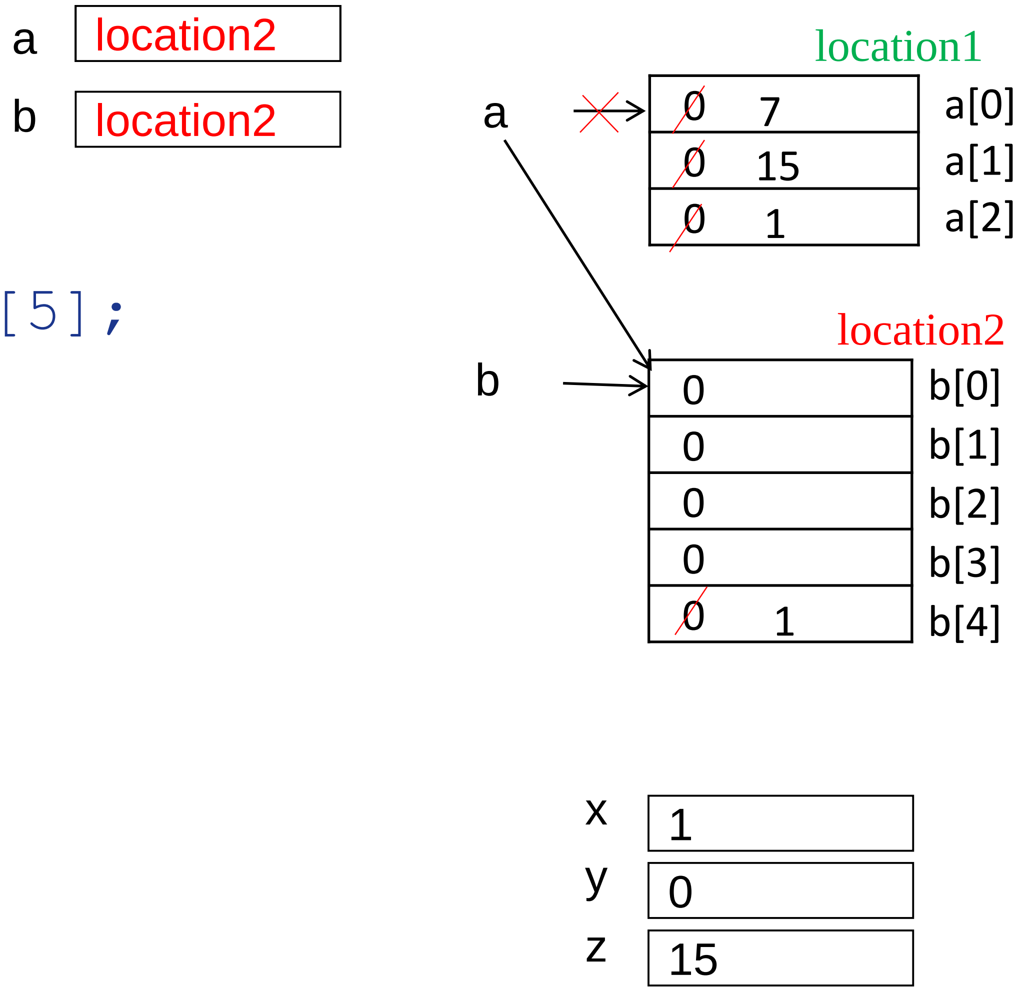
```
int[] a ;
a = new int[3];

int[] b = new int[5];

a[0] = 7;
a[1] = a[0] + 8;
a[2]++;

int x = a[2];
int y = a[2] - 1;
int z = a[2-1];

a = b;
a[a.length-1]++;
```



Clicker Question

What is the output?

A. 3 references, 3 objects

B. 3 references, 2 objects

C. 3 references, 1 object

D. 2 references, 2 objects

E. Error

```
int[] a = new int[6];
int[] b = new int[4];
int[] c = new int[6];
a = b;
a = c;
for(int i=0; i<4; i++)
   c[i] = a[i];
```

Clicker Question

How many array references and objects do we have at the end of this code?

A. 3 references, 3 objects

B. 3 references, 2 objects

C. 3 references, 1 object

D. 2 references, 2 objects

E. Error

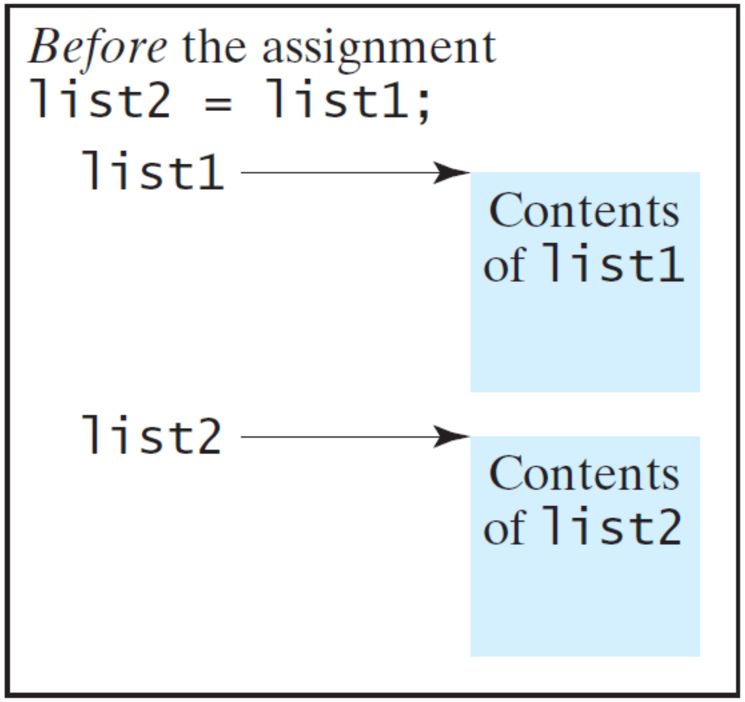
```
int[] a = new int[6];
int[] b = {1,2,3,4,5};
int[] c = b;
for(int i=0; i<4; i++)
   c[i] = a[i];
a = b;
```

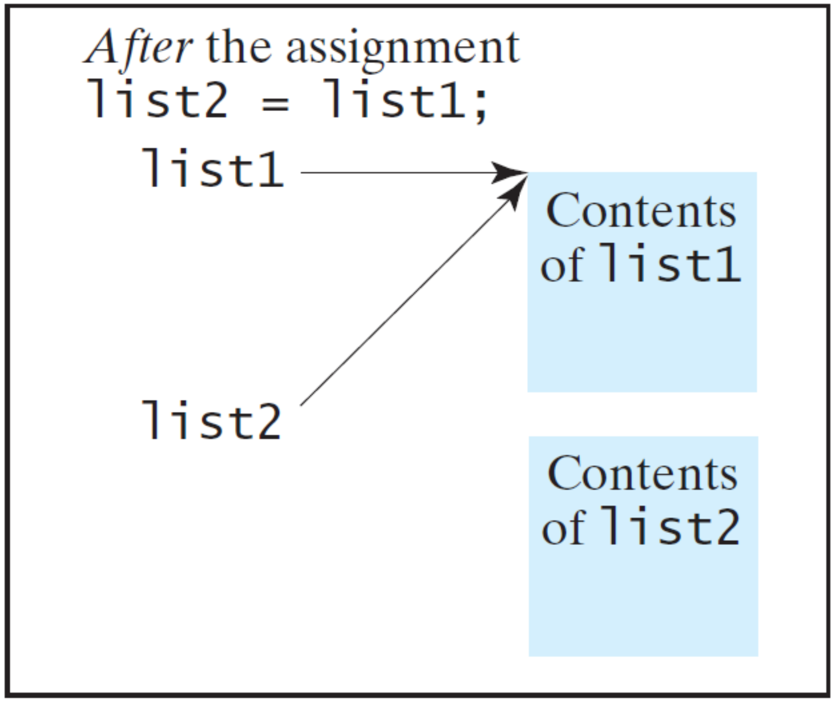
Copying Arrays

Copying Arrays

Given two arrays list1 and list 2, the statement, **`list2 = list1;`** **does not copy the contents** of **list1** to **list2**, but instead merely copies the reference value from **list1** to **list2**

- *garbage collection:* An array that has no reference becomes garbage, which will be collected by the Java Virtual Machine.





Copying Arrays, cont.

You may use of the following ways to copy arrays:

1. Use a loop to **copy individual elements** one by one.
2. Use the **`arraycopy`** method in the **System** class.

   Syntax of arraycopy is as follows:

   arraycopy(sourceArray, srcPos, targetArray, tarPos, length);

Example: consider the following two arrays:

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

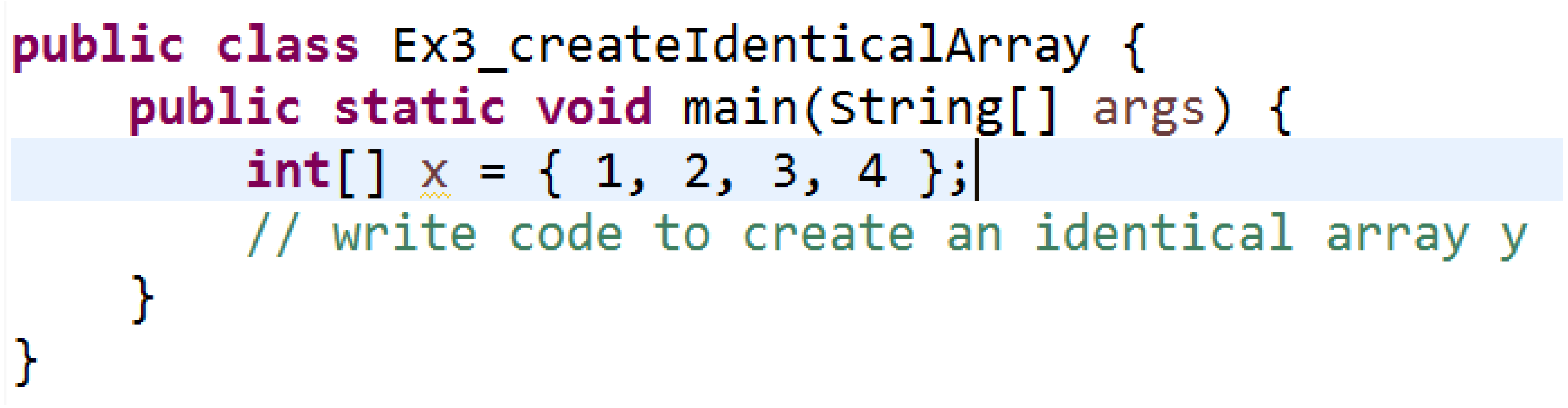
(1) copying using a loop:

```
for (int i = 0; i < source.length; i++)
   target[i] = source[i];
```

(2) copying using arraycopy:

```
System.arraycopy(source, 0, target, 0, source.length);
```

Practice



```java
public class Ex3_createIdenticalArray {
    public static void main(String[] args) {
        int[] x = { 1, 2, 3, 4 };
        // write code to create an identical array y
    }
}
```

For Each Loops

For-each Loops

for-each loops enable you to traverse the complete array sequentially **without using an index variable**.

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

Example: display all elements in myList:

```
for (int item: myList)
      System.out.println(item);
```

Clicker Question

What are the contents of the two arrays x and y?

```
int[] x = {1,1,1}, y = {1,1,1};
for(int item: x)
   item++;
for(int i = 0; i < y.length; i++)
   y[i]++;
```

A. x: 1,1,1 y: 1,1,1

B. x: 2,2,2 y: 2,2,2

C. x: 1,1,1 y: 2,2,2

D. x: 1,2,3 y: 2,2,2

Arrays and Methods

Passing Arrays to Methods

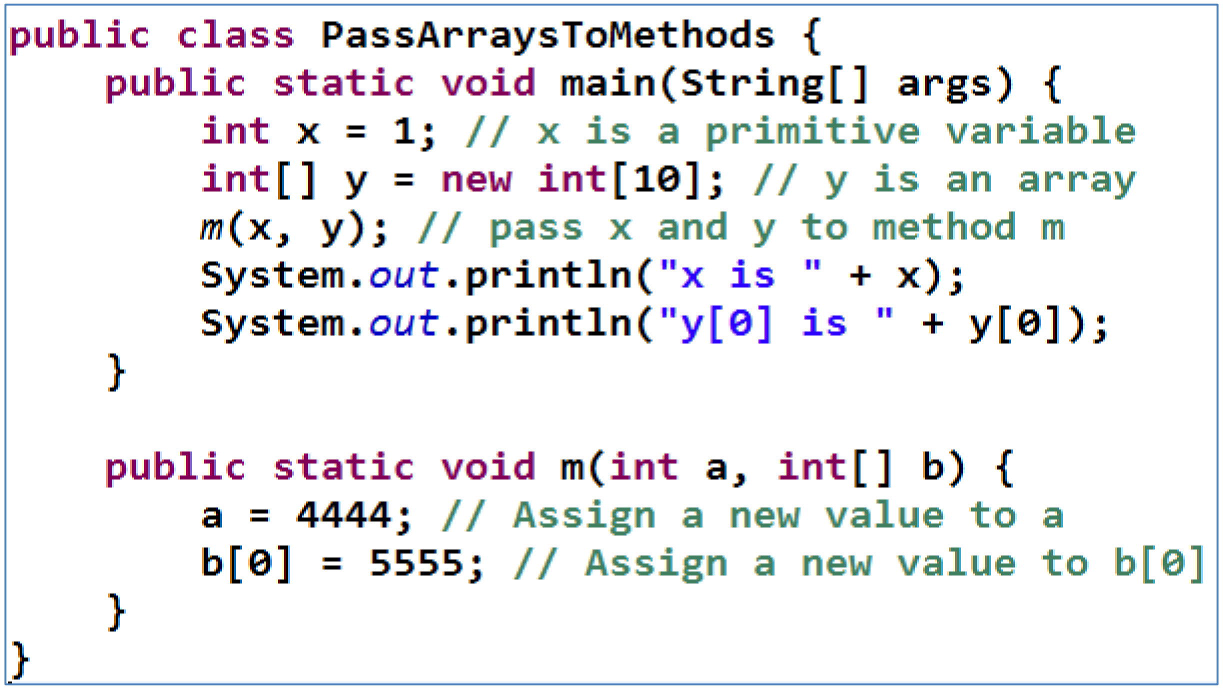
Java uses **pass-by-value** to pass arguments to a method.

If the parameter is of:

- a primitive type:
  - 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.
- 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.
  - **The array in the method is the SAME AS the array being passed**.

Passing Arrays to Methods, cont.

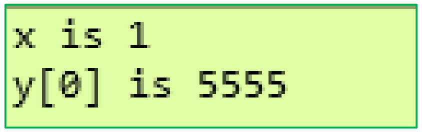
In the following example, x will not change within the method, but y will change. Can you explain why?



```java
public class PassArraysToMethods {
    public static void main(String[] args) {
        int x = 1; // x is a primitive variable
        int[] y = new int[10]; // y is an array
        m(x, y); // pass x and y to method m
        System.out.println("x is " + x);
        System.out.println("y[0] is " + y[0]);
    }

    public static void m(int a, int[] b) {
        a = 4444; // Assign a new value to a
        b[0] = 5555; // Assign a new value to b[0]
    }
}
```

Output:



```
x is 1
y[0] is 5555
```

Practice

Write two methods that return the minimum and maximum elements of an array with the following headers:

public static int min(**int**[] ar)

public static int max(**int**[] ar)

Test your methods with appropriate data.

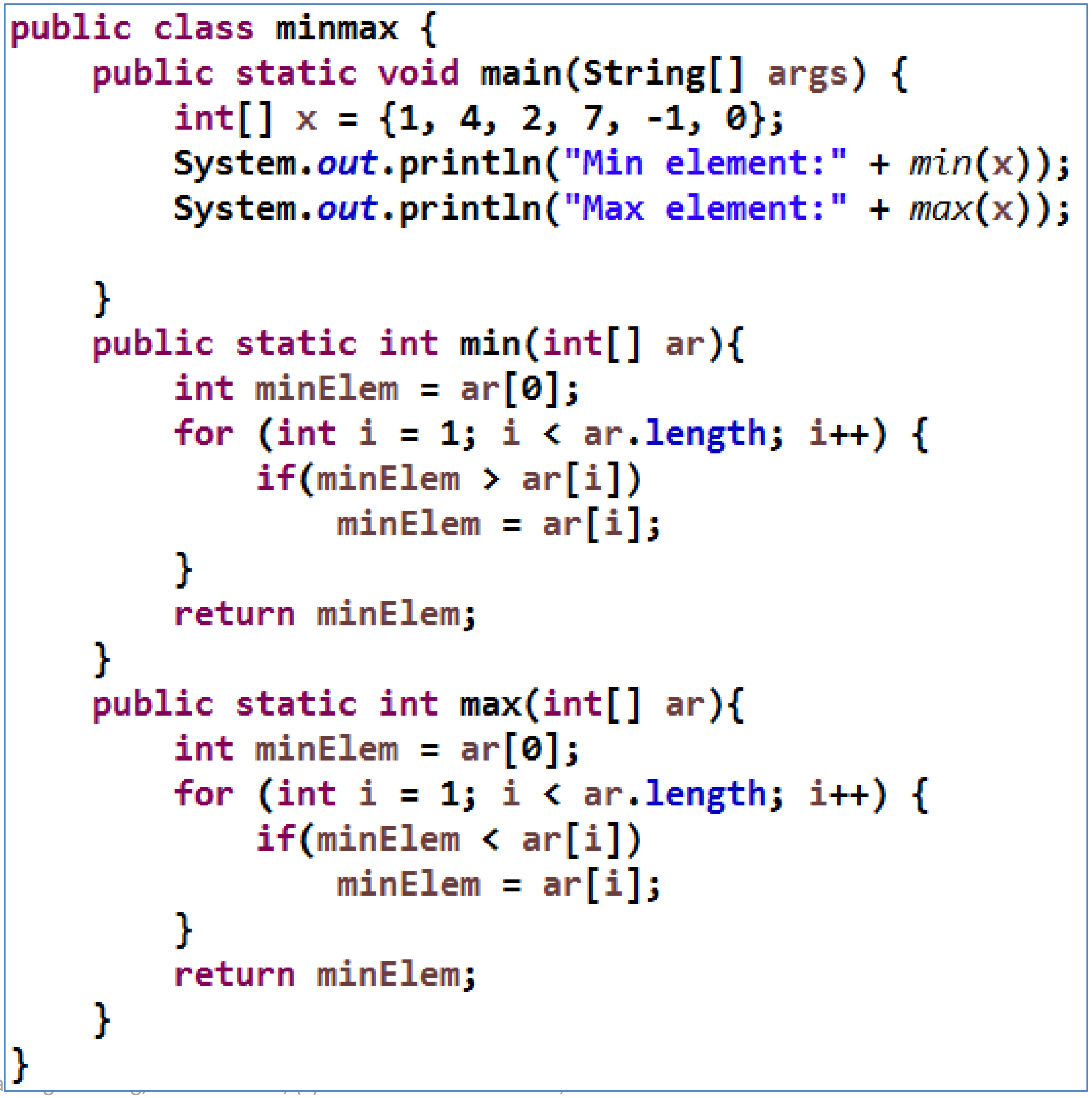
Basic idea for min:

- Declare a variable *minElem* and assign the first array element to it.
- Compare *minElem* to the remaining elements of the array. Whenever an element has less value, store that value in *min*.

Basic idea for max:

Similar to that of min except that we find the maximum element.

Practice, cont.



```
public class minmax {
    public static void main(String[] args) {
        int[] x = {1, 4, 2, 7, -1, 0};
        System.out.println("Min element:" + min(x));
        System.out.println("Max element:" + max(x));

    }
    public static int min(int[] ar){
        int minElem = ar[0];
        for (int i = 1; i < ar.length; i++) {
            if(minElem > ar[i])
                minElem = ar[i];
        }
        return minElem;
    }
    public static int max(int[] ar){
        int minElem = ar[0];
        for (int i = 1; i < ar.length; i++) {
            if(minElem < ar[i])
                minElem = ar[i];
        }
        return minElem;
    }
}
```

Practice, cont.

1) What happens if we change the values of ar[i] within any of the two methods?

2) What should be changed in the methods if we want to find the min and max of the array of double values {1.3, 1.8, 2.4, 4.2} ?

3) How to have two methods for min (and two for max), one to handle integers and the other for doubles?

Returning an Array from a Method

Based on the discussion in this section, when a method returns an array **the reference of the array is returned**.

Sample method that returns an array:

```
Public void int[] amethod(){
    int[] x = {1, 2, 4};
    return x;
}
```

Practice

Write a method that has the following header:

public static int[] getRandomArray(int n, int from, int to)

The method should return an array of *n* random numbers from *from* to *to*.

Test your program by calling the method and printing out its elements.

Remember:

`a + Math.random() * b` $\longrightarrow$ Returns a random number between `a` and `a + b`, excluding `a + b`.

Practice

Write a method that

 * receives one input: radius

 * returns two results: area and perimeter

The idea:

- a method can only return one item, but this item can be an object– objects may contain several values (e.g. arrays)
- in the question above, you can return the two results as a 2-element array.

- Anonymous Array
- Variable-Length Argument Lists

Anonymous Array

Anonymous arrays have no explicit reference variable.

They are created and and initialized in the same line

```
new dataType[]{literal0,literal1,…,literalk}
```

The must be used as part of another statement. For example,

- As an argument

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

- In a for-each loop

  ```
  for (int item: new int[]{3, 1, 2, 6} )
           System.out.println(item);
  ```

Variable-Length Argument Lists

You can pass a variable number of arguments of the same type to a method.

```
returnType amethod (type... parameterName)
```

Some rules…

- **Only one variable-length parameter** may be specified.
- This parameter must **be the last parameter**.
- You can **pass** an **array** OR a **variable number of arguments** to a variable-length parameter.

Java treats a variable-length parameter as an array.

Variable-Length Argument Lists, cont.

```java
public class VarArgsDemo {
    public static void main(String[] args) {
        printMax(34, 3, 3, 2, 56.5);
        printMax(new double[] { 1, 2, 3 });
    }

    public static void printMax(double... numbers) {
        if (numbers.length == 0) {
            System.out.println("No argument passed");
            return;
        }
        double result = numbers[0];
        for (int i = 1; i < numbers.length; i++)
            if (numbers[i] > result)
                result = numbers[i];
        System.out.println("The max value is " + result);
    }
}
```

Output:

```
The max value is 56.5
The max value is 3.0
```

Practice

Write a method with this header and test it.

void printRecord(String name, int age, int... grades)

The **Arrays** Class

The **Arrays** Class

The **java.util.Arrays** class contains useful methods for common array operations.

Consider two arrays int[] myList and int[] secondList

```
void sort(myList)
```

- Sorts myList into **ascending** numerical order.

```
int binarySearch(myList, value)
```

- Searches myList for the specified value and returns its index.
- The array **must be sorted** prior to making this call.

```
String toString(myList)
```

- Returns a string representation of the contents of myList.

```
Boolean equals(myList, secondList)
```

- Returns **true** if the two arrays are strictly equal, and returns false otherwise. Two arrays are strictly equal if their corresponding elements are the same.

The Arrays Class, cont.

```java
public static void main(String[] args) {
    int myList[] = {4, 1, 5, 11, 9, 3, 10, 8, 2};
    int secondList[] = {4, 1, 5, 11, 9, 3, 10, 8, 2};
    if(Arrays.equals(myList, secondList))
        System.out.println("myList is strictly equal to secondList.");
    else
        System.out.println("myList is not strictly equal to secondList");
    System.out.print("myList prior sorting: ");
    System.out.println(Arrays.toString(myList));
    Arrays.sort(myList);
    System.out.print("myList after sorting: ");
    System.out.println(Arrays.toString(myList));
    System.out.print("The index of number 11 in the sorted myList: ");
    System.out.println(Arrays.binarySearch(myList, 11));
}
```

Output:

```
myList is strictly equal to secondList.
myList prior sorting: [4, 1, 5, 11, 9, 3, 10, 8, 2]
myList after sorting: [1, 2, 3, 4, 5, 8, 9, 10, 11]
The index of number 11 in the sorted myList: 8
```