

COMP30820
Java Programming (Conv)

Michael O'Mahony

Chapter 7 Single-Dimensional Arrays

Objectives

- To declare array reference variables and create arrays (§§7.2.1–7.2.2).
- To obtain array size and know the default values in an array (§7.2.3).
- To access array elements using indices (§7.2.4).
- To declare, create, and initialize an array using an array initializer (§7.2.5).
- To program common array operations (displaying arrays, summing all elements, finding the minimum and maximum elements) (§7.2.6).
- To simplify programming using **foreach** loops (§7.2.7).
- To copy contents from one array to another (§7.5).
- To develop and invoke methods with array arguments and return values (§§7.6–7.8).
- To use the methods in the **java.util.Arrays** class (§7.12).

Introducing Arrays

An array is a data structure that stores a fixed-size, sequential collection of elements of the same type.

Example – an array (`myList`) with 10 elements of type `double`:

	<code>myList[0]</code>	2.1	
	<code>myList[1]</code>	3.2	
	<code>myList[2]</code>	8.1	
	<code>myList[3]</code>	0.4	
	<code>myList[4]</code>	0.9	
Array element at index 5	<code>myList[5]</code>	1.3	← Element value
	<code>myList[6]</code>	3.2	
	<code>myList[7]</code>	4.8	
	<code>myList[8]</code>	6.4	
	<code>myList[9]</code>	7.2	

Declaring and Creating Arrays

- Syntax:

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

Example:

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

The Length of an Array

When an array is created, the array size must be given, specifying the number of elements that can be stored in it.

Once an array is created, its size is fixed and cannot be changed.

You can find the *size of* (aka *length of*, aka *number of elements in*) an array using:

```
arrayRefVar.length
```

Example:

```
double[] myList = new double[10];  
int len = myList.length // len is 10
```

Default Values

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

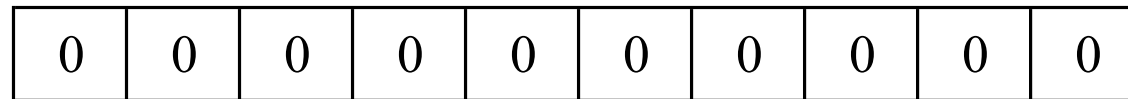
- 0 for the numeric primitive data types
- `'\u0000'` for `char` types
- `false` for `boolean` types

Indexed Variables

Array elements are accessed through *indices*. For example:

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

indices 0 1 2 3 4 5 6 7 8 9



myList[0]

myList[9]

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

```
arrayRefVar[index];
```

For example: myList[0], myList[1], ...

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 assigns values to `myList[0]` and `myList[1]`:

```
myList[0] = 2.1;  
myList[1] = 3.2;
```

And the following adds the values in `myList[0]` and `myList[1]` and assigns the result to `myList[2]`:

```
myList[2] = myList[0] + myList[1]; // myList[2] is 5.3
```

Declaring, Creating, Initializing

Declaring, creating, initializing in one step:

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

This shorthand notation 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;
```

Note

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

Splitting it causes a syntax error. For example, the following is incorrect:

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


Trace Program with Arrays

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[3];  
  
        for (int i = 0; i < values.length; i++)  
            values[i] = i + 1;  
    }  
}
```

Trace Program with Arrays

Declare and create an array values

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[3];  
  
        for (int i = 0; i < values.length; i++)  
            values[i] = i + 1;  
    }  
}
```



0	0
1	0
2	0

Trace Program with Arrays

Initialize `i` to 0

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[3];  
  
        for (int i = 0; i < values.length; i++)  
            values[i] = i + 1;  
    }  
}
```

0	0
1	0
2	0

Trace Program with Arrays

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[3];  
  
        for (int i = 0; i < values.length; i++)  
            values[i] = i + 1;  
    }  
}
```

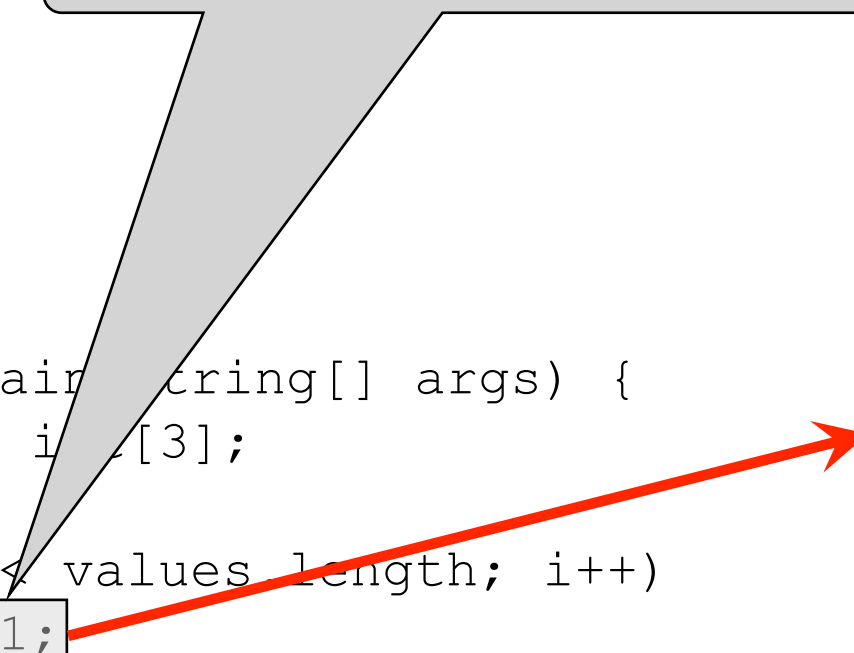
0 is less than 3

0	0
1	0
2	0

Trace Program with Arrays

After this statement, `values[0]` is 1 ($0 + 1$)

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[3];  
  
        for (int i = 0; i < values.length; i++)  
            values[i] = i + 1;  
    }  
}
```



0	1
1	0
2	0

Trace Program with Arrays

Increment *i* (*i* is now 1)

```
public class Test {  
    public static void main(String[] args)  
        int[] values = new int[3];  
  
    for (int i = 0; i < values.length; i++)  
        values[i] = i + 1;  
}
```

0	1
1	0
2	0

Trace Program with Arrays

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[3];  
  
        for (int i = 0; i < values.length; i++)  
            values[i] = i + 1;  
    }  
}
```

1 is less than 3

0	1
1	0
2	0

Trace Program with Arrays

After this statement, `values[1]` is 2 ($1 + 1$)

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[3];  
  
        for (int i = 0; i < values.length; i++)  
            values[i] = i + 1;  
    }  
}
```

0	1
1	2
2	0

Trace Program with Arrays

Increment `i` (`i` is now 2)

```
public class Test {  
    public static void main(String[] args)  
        int[] values = new int[3];  
  
    for (int i = 0; i < values.length; i++)  
        values[i] = i + 1;  
}
```

0	1
1	2
2	0

Trace Program with Arrays

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[3];  
  
        for (int i = 0; i < values.length; i++)  
            values[i] = i + 1;  
    }  
}
```

2 is less than 3

0	1
1	2
2	0

Trace Program with Arrays

After this statement, `values[2]` is 3 ($2 + 1$)

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[3];  
  
        for (int i = 0; i < values.length; i++)  
            values[i] = i + 1;  
    }  
}
```

0	1
1	2
2	3

Trace Program with Arrays

Increment `i` (`i` is now 3)

```
public class Test {  
    public static void main(String[] args)  
        int[] values = new int[3];  
  
    for (int i = 0; i < values.length; i++)  
        values[i] = i + 1;  
}
```

0	1
1	2
2	3

Trace Program with Arrays

3 is not less than 3, exit loop

```
public class Test {  
    public static void main(String[] args) {  
        int[] values = new int[3];  
  
        for (int i = 0; i < values.length; i++)  
            values[i] = i + 1;  
    }  
}
```

0	1
1	2
2	3

Processing Arrays

Some code snippets to show examples of using arrays:

1. Initializing arrays with input values
2. Printing arrays
3. Summing all elements
4. Finding the largest element
5. Finding the index of the largest element

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();
```

Printing arrays

```
double[] myList = new double[10];  
  
for (int i = 0; i < myList.length; i++)  
    System.out.print(myList[i] + " ");
```

Summing all elements

```
double[] myList = new double[10];  
double sum = 0;  
  
for (int i = 0; i < myList.length; i++)  
    sum += myList[i];
```

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];
```

Finding the index of the largest element

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

foreach Loop

foreach loop – enables you to traverse the complete array sequentially without using an index variable.

The following are equivalent and display all elements in the array:

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

foreach Loop

foreach loop – enables you to traverse the complete array sequentially without using an index variable.

The following are equivalent and display all elements in the array:

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

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


foreach Loop

foreach loop – enables you to traverse the complete array sequentially without using an index variable.

The following are equivalent and display all elements in the array:

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

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

```
for (double x: myList)  
    System.out.print(x + " ");
```

foreach Loop

foreach loop – enables you to traverse the complete array sequentially without using an index variable.

The following are equivalent and display all elements in the array:

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

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

```
for (double x: myList)  
    System.out.print(x + " ");
```

General syntax:

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

Note: you need to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

Problem: Lotto Numbers

Suppose you play *Pick-10* lotto. Each ticket has 10 numbers ranging from 1 to 99, inclusive.

Assume that the numbers in a ticket are picked randomly. Further, assume that the same number may appear in a ticket more than once.

Over all your tickets, you wish to have all the numbers from 1 to 99 included at least once.

Write a program that generates lotto tickets. Print the tickets to the standard output. The program should exit when all numbers from 1 to 99 are included at least once over all tickets, and the number of tickets generated should be displayed.

LottoTickets

Problem: Lotto Numbers

isCovered

[0]	false
[1]	false
[2]	false
[3]	false
	.
	.
	.
[97]	false
[98]	false

(a)

isCovered

[0]	true
[1]	false
[2]	false
[3]	false
	.
	.
	.
[97]	false
[98]	false

(b)

isCovered

[0]	true
[1]	true
[2]	false
[3]	false
	.
	.
	.
[97]	false
[98]	false

(c)

isCovered

[0]	true
[1]	true
[2]	true
[3]	false
	.
	.
	.
[97]	false
[98]	false

(d)

isCovered

[0]	true
[1]	true
[2]	true
[3]	false
	.
	.
	.
[97]	false
[98]	true

(e)

Reference Types

Declaring array variables:

- Consider:

```
int[] arr;
```

- An array variable (e.g. `arr`) is a *reference variable* – when one is declared, it does not allocate space in memory for an array, it is a variable that contains a reference to an array.

Creating arrays:

- An array is created by using the `new` operator and its reference is assigned to the variable `arr` using the following syntax:

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

- The above statement does the following:
 - 1) It creates an array using `new int[3];`
 - 2) It assigns the reference of the newly created array to the variable `arr`.
- You cannot assign elements to an array unless it has already been created.

Reference Types

Suppose you created two arrays as follows:

```
int[] list1 = {1, 2, 3, 4, 5};  
int[] list2 = {10, 20, 30};
```

What happens after the following statements?

```
list2 = list1;  
list1[0] = 100;  
list2[1] = 200;
```

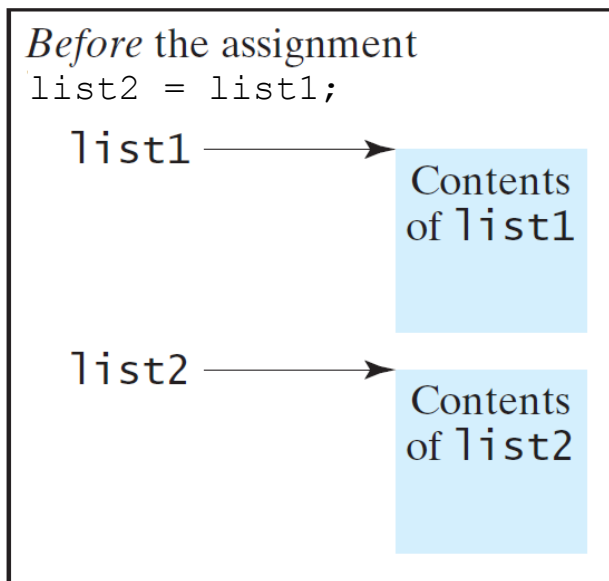
Reference Types

Suppose you created two arrays as follows:

```
int[] list1 = {1, 2, 3, 4, 5};  
int[] list2 = {10, 20, 30};
```

What happens after the following statements?

```
list2 = list1;  
list1[0] = 100;  
list2[1] = 200;
```



Reference Types

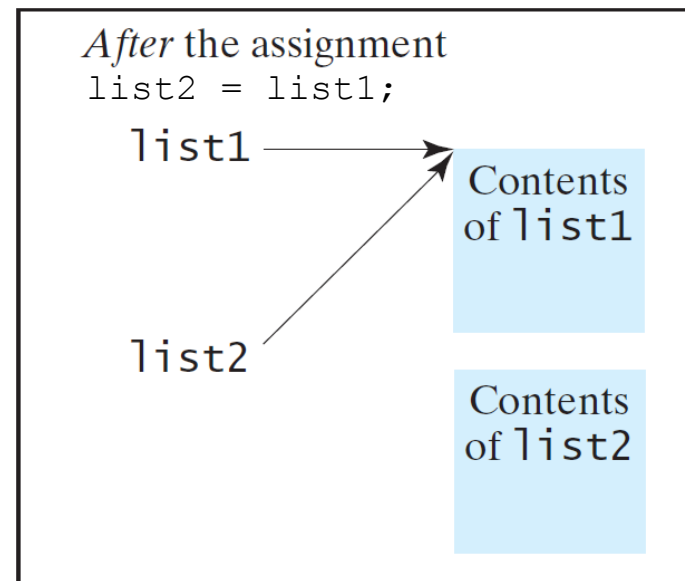
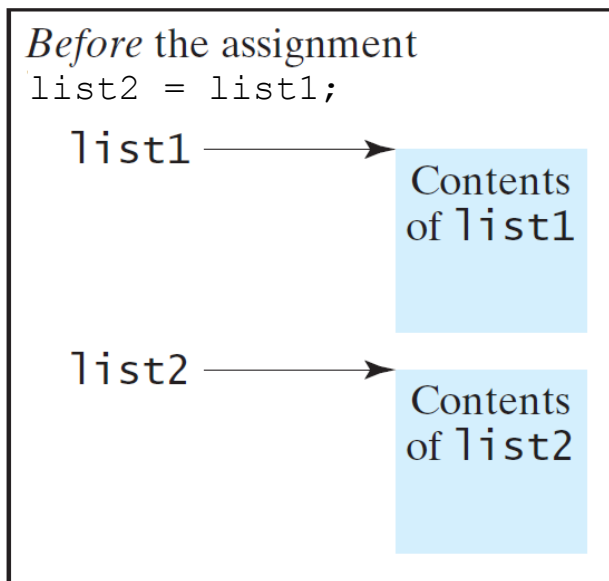
Suppose you created two arrays as follows:

ArrayReferenceTest

```
int[] list1 = {1, 2, 3, 4, 5};  
int[] list2 = {10, 20, 30};
```

What happens after the following statements?

```
list2 = list1;  
list1[0] = 100;  
list2[1] = 200;
```



Passing Arrays to Methods

When passing an array to a method, the *reference* of the array is passed to the method.

Example – a method to print an array:

```
public static void main(String[] args) {  
    int[] list = {3, 1, 2, 6, 4, 2};  
    printArray(list); // Invoke the method  
}
```

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

Pass By Value

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

There are important differences between passing primitive data type variables and passing array variables to methods...

For a parameter of a primitive type, changing the value of the parameter inside the method does not affect the value of the variable passed to the method (we have seen this before...).

For a parameter of an array type, any changes made to the array elements inside the method will affect the original array that was passed to the method.

Passing Arrays as Arguments

Examples to demonstrate the differences between passing primitive data type variables and array variables to methods.

TestPassPrimitive

TestPassArray1

TestPassArray2

Returning an Array from a Method

When a method returns an array, the *reference* of the array is returned.

For example:

```
public static int[] createArray(int size) {  
    int[] arr = new int[size];  
  
    for (int i = 0; i < arr.length; i++)  
        arr[i] = i + 1;  
  
    return arr;  
}
```

Returning an Array from a Method

When a method returns an array, the *reference* of the array is returned.
For example:

```
public static int[] createArray(int size) {  
    int[] arr = new int[size];  
  
    for (int i = 0; i < arr.length; i++)  
        arr[i] = i + 1;  
  
    return arr;  
}
```

Invoke this method in another part of the program (for example, in the `main` method):

```
int[] nums = createArray(4); // [1, 2, 3, 4]
```

Problem: Counting the Occurrence of Each Letter

Randomly generate 20 lowercase letters and assign to an array of characters.

Count the occurrence of each letter in the array and display the result.

CountLettersInArray

Copying Arrays

To copy the contents of one array into another, you have to copy the array's individual elements into the other array.

```
int[] sourceArray = {2, 3, 1, 5, 10};
```

```
int[] targetArray = new int[sourceArray.length];
```

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

The `Arrays.sort` Method

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of `byte`, `short`, `int`, `long`, `float`, `double`, and `char` in the `java.util.Arrays` class.

The `Arrays.sort` Method

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of `byte`, `short`, `int`, `long`, `float`, `double`, and `char` in the `java.util.Arrays` class.

For example, the following code sorts (1) an array of numbers and (2) an array of characters.

The `Arrays.sort` Method

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of `byte`, `short`, `int`, `long`, `float`, `double`, and `char` in the `java.util.Arrays` class.

For example, the following code sorts (1) an array of numbers and (2) an array of characters.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};  
java.util.Arrays.sort(numbers);
```

The `Arrays.sort` Method

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of `byte`, `short`, `int`, `long`, `float`, `double`, and `char` in the `java.util.Arrays` class.

For example, the following code sorts (1) an array of numbers and (2) an array of characters.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};  
java.util.Arrays.sort(numbers);  
// result: {1.9, 2.9, 3.4, 3.5, 4.4, 6.0}
```

The `Arrays.sort` Method

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of `byte`, `short`, `int`, `long`, `float`, `double`, and `char` in the `java.util.Arrays` class.

For example, the following code sorts (1) an array of numbers and (2) an array of characters.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};  
java.util.Arrays.sort(numbers);  
// result: {1.9, 2.9, 3.4, 3.5, 4.4, 6.0}
```

```
char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};  
java.util.Arrays.sort(chars);
```

The `Arrays.sort` Method

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of `byte`, `short`, `int`, `long`, `float`, `double`, and `char` in the `java.util.Arrays` class.

For example, the following code sorts (1) an array of numbers and (2) an array of characters.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};  
java.util.Arrays.sort(numbers);  
// result: {1.9, 2.9, 3.4, 3.5, 4.4, 6.0}
```

```
char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};  
java.util.Arrays.sort(chars);  
// result: {'4', 'A', 'D', 'F', 'P', 'a'}
```

ASCII Code for Commonly Used Characters

Characters	Code Value in Decimal	Unicode Value
'0' to '9'	48 to 57	\u0030 to \u0039
'A' to 'Z'	65 to 90	\u0041 to \u005A
'a' to 'z'	97 to 122	\u0061 to \u007A

The `Arrays.toString` Method

The `java.util.Arrays.toString` method can be used to return a string representation for the array.

For example:

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};  
System.out.println(java.util.Arrays.toString(numbers));  
  
// Prints: [6.0, 4.4, 1.9, 2.9, 3.4, 3.5]
```

Next Topic

Chapter 8:

- Multidimensional arrays...