

Chapter 7:

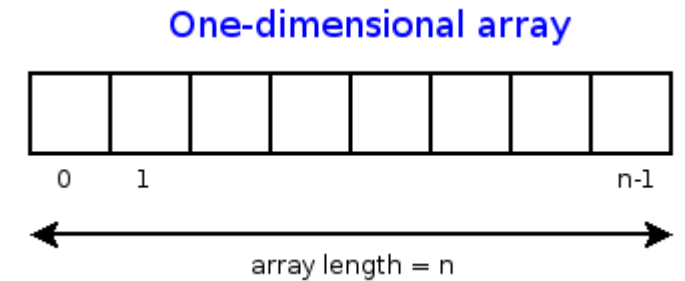
Arrays

WEEK 9:

ARRAY BASICS

ARRAY TRAVERSAL

MULTIDIMENSIONAL ARRAYS



Array Basics

- **Array:** An indexed structure that holds multiple values of the SAME type
- **Index:** An integer indicating the position of a particular value in a data structure
- **Zero-based Indexing:** A numbering scheme used throughout Java in which a sequence of values is indexed starting with zero (element 0, element 1, element 2, etc.)
- To construct an array, you need to specify both the **TYPE** of elements you want to store in the array and the length of the array (**NUMBER** of elements it can hold)
- **General Syntax:**

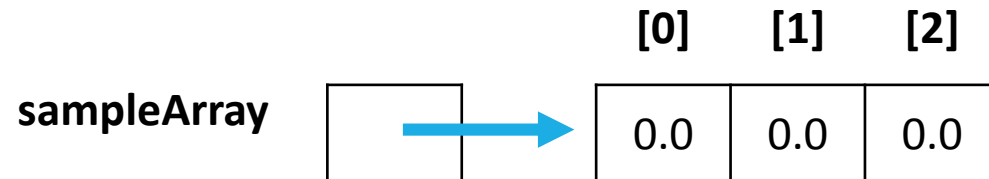
```
<element type>[] <name> = new <element type>[<length>];
```

Constructing an Array

- Consider the following statement that constructs an array of three doubles:

```
double[] sampleArray = new double[3];
```

- The variable `sampleArray` is not the array itself – it stores a REFERENCE to the array as shown



Type	Value
int	0
double	0.0
char	'\0'
boolean	false
objects	null

- When an array is constructed, Java **auto-initializes** each element to a default value, as shown in the table to the right

Storing Values in an Array

- To store new values in an array (i.e. to overwrite the default values), you reference a location in the array by using its index:

sampleArray[0] = 103.78;

- Values can also be stored in an array by using a loop. For example, we can fill an array with random numbers using the following code:
- Note: When determining the length of an array, just use **arrayName.length** (*without* parentheses after the word “length”)

```
public class ArrayTest {  
    public static void main(String[] args) {  
        int[] sampleArray = new int[10];  
        for(int i = 0; i < sampleArray.length; i++){  
            sampleArray[i] = (int)(Math.random()*100);  
        }  
  
        for(int i = 0; i < sampleArray.length; i++){  
            System.out.print "[" + sampleArray[i] + " " );  
        }  
        System.out.println();  
    }  
}
```

Console

```
<terminated> ArrayTest [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe  
[81][73][32][7][11][80][43][94][21][74]
```

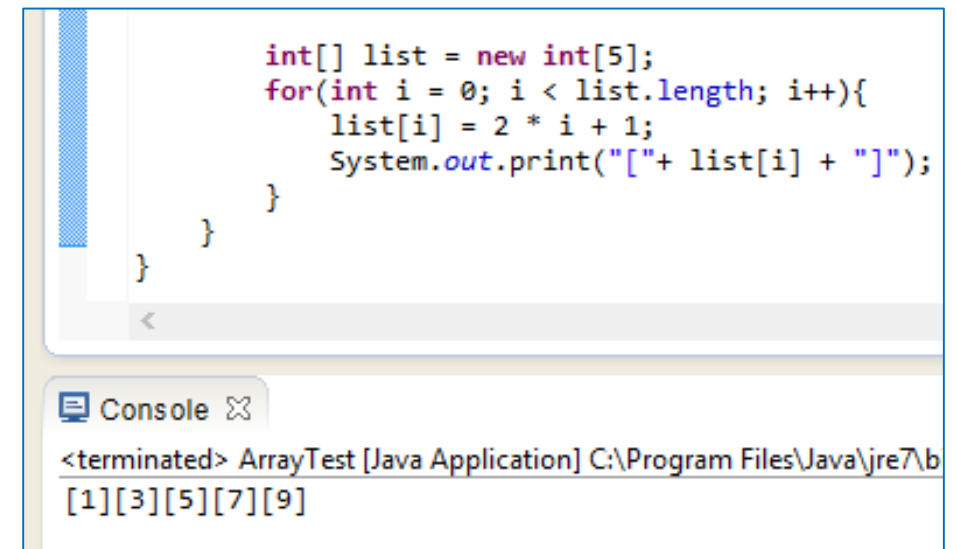
Array Traversal

- On the last slide, we used a for loop to replace the values stored in the array
- This strategy is very useful any time you want to traverse through the elements in an array
- **Array Traversal:** Processing each array element sequentially from first to last
- **General Form:**

```
for (int i = 0; i < array.length; i++){  
    <do something with array[i]>;  
}
```

- **Example:**

This example uses a for loop to both assign values to the array and to print out the array.



The screenshot shows a Java IDE with a code editor and a console window. The code in the editor is a Java program that creates an integer array of size 5, iterates through it with a for loop, and prints each element. The console output shows the array elements: [1][3][5][7][9].

```
int[] list = new int[5];  
for(int i = 0; i < list.length; i++){  
    list[i] = 2 * i + 1;  
    System.out.print "[" + list[i] + "];"  
}  
}
```

Console

<terminated> ArrayTest [Java Application] C:\Program Files\Java\jre7\b
[1][3][5][7][9]

ArrayIndexOutOfBoundsException



- What happens if you try in process data from an index that does not exist in your array?
- Not surprisingly, Java will halt your program and inform you that you have an error in your program
- Specifically, your program has thrown an `ArrayIndexOutOfBoundsException`
- In the sample code, trying to access the index “`list.length`” caused the error – this emphasizes the fact that indexing starts at **ZERO**
- An array of length 5, like our array, will have indices numbered 0, 1, 2, 3, and 4

```
int[] list = new int[5];
for(int i = 0; i < list.length; i++){
    list[i] = 2 * i + 1;
    System.out.print "[" + list[i] + " ";
}
System.out.println();
System.out.println(list[list.length]);
```

Console

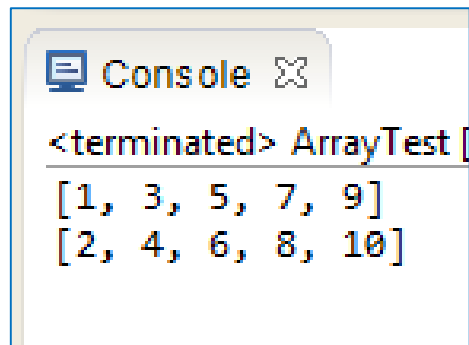
<terminated> ArrayTest [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 1, 2015, [1][3][5][7][9]
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
at ArrayTest.main(ArrayTest.java:24)

Random Access

- We have been using primarily **sequential access** to process data in our programs
- **Sequential Access:** Manipulating values in a sequential manner from first to last
- Arrays provide us with a more flexible way to access data – **random access**
- **Random Access:** Manipulating values in any order whatsoever to allow quick access to each value
- An array can provide random access because it is allocated as a CONTINUOUS BLOCK in memory
- As a result, the computer can quickly compute exactly where a particular value will be located

Arrays and Methods

- When you pass an array to a method, the method has the ability to **CHANGE THE CONTENTS** of the array – even if the method does NOT return anything to the calling method.
- Look at the code at the right.
- The output of this code is shown below:



```
Console
<terminated> ArrayTest [
[1, 3, 5, 7, 9]
[2, 4, 6, 8, 10]
```

```
public class ArrayTest {

    public static void main(String[] args) {
        // create array of length 5
        int[] list = new int[5];
        // fill array with odd integers
        for(int i = 0; i < list.length; i++){
            list[i] = 2 * i + 1;
        }
        // print array
        System.out.print("[ " + list[0]);
        for(int i = 1; i < list.length; i++){
            System.out.print(", " + list[i]);
        }
        System.out.print("]\n");
        // call incrementAll method
        incrementAll(list);
        // print array again
        System.out.print("[ " + list[0]);
        for(int i = 1; i < list.length; i++){
            System.out.print(", " + list[i]);
        }
        System.out.print("]\n");
    }

    public static void incrementAll(int[] inputArray){
        for(int i = 0; i < inputArray.length; i++){
            inputArray[i]++;
        }
    }
}
```


Methods that Return Arrays

- What if you want to write a method that, instead of taking an array as an input parameter, creates an array that it sends back to the calling method
- To do this, you just need to specify in the method header that the return type of the method will be some type of array

```
public static int[] fillWithOdds(int n){  
    int[] newArray = new int[n];  
    for(int i = 0; i < n; i++){  
        newArray[i] = 2 * i + 1;  
    }  
    return newArray;  
}
```

The for-each Loop

- Java has a variation of the for loop that is especially handy for traversing an array
- This loop is known as an “**enhanced for loop**” or a “**for-each loop**”
- To traverse an array of odd numbers, **odds**, to see if the array stores the value 11, we could use the following code:

```
for (int each: newArray){  
    if (each == 11){  
        System.out.println("\nFound the value 11 in the array!");  
        //<or whatever else we want done once the target value is found>  
    }  
}
```

This code goes through the array index by index, sequentially storing every index's value in “each” – the simple if statement in the loop body checks each value and performs the controlled block of code.

Initializing Arrays

- Java provides a useful shortcut for initializing values of an array
- Rather than entering each value individually in a separate command, the values may all be entered using a single statement
- For example, you might want to store the number of days in each month in an array, so that these values could be looked up quickly:

```
int[] monthDays = {29, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
```

- In this array, the number of days in month 1 (January) is stored at index 1 (index 0 holds the number of days in Leap Year February)
- General syntax:

```
<element type>[] <name> = {<value>, <value>, . . ., <value>;}
```

The Arrays Class

- Java has a built-in Arrays Class that attempts to make arrays easier to work with (to use it, you would need to include an import declaration – it is in the java.util package)
- One of the limitations of arrays is that you can't change the size of an array in the middle of program execution (remember: arrays are allocated as a contiguous block of memory when they are created)
- To change the size of an array you are working with, you actually need to create a new array and then copy over the values from the old array to the new array
- Using the Arrays class, you could create a new array that is twice the size of your original array and that contains the same data as follows:

```
int[] newData = Arrays.copyOf(data, 2 * data.length)
```

Useful Arrays Class Methods

Method	Description
<code>copyOf(array, newSize)</code>	Returns a copy of the array with the given size
<code>copyOfRange(array, startIndex, endIndex)</code>	Returns a copy of the given subportion of the given array
<code>equals(array1, array2)</code>	Returns true if the arrays contain the same elements
<code>fill(array, value)</code>	Sets every element of the array to the given value
<code>sort(array)</code>	Rearranges the elements so that they appear in sorted (increasing) order
<code>toString(array)</code>	Returns a String representation of the array, as in [3, 5, 7]

NOTE: the *toString* method makes the array easier to print. Remember we had needed to use a for loop to print out arrays. This is because the variable naming the array hold only a REFERENCE to the array, not the array itself.

Array-Traversal Algorithms

- Previous slides demonstrated how to traverse an array using a for loop or a for-each loop.
- There are some tasks that are frequently performed on arrays, such as searching for a specific value in an array or counting the number of times a value appears in an array
- The following code fragments show one way to accomplish each of these tasks:

```
int target = 5;
int count = 0;
int[] list = {5, 3, 6, 4, 9, 5, 5, 89, 49, 87, 5, 93};

for (int n: list){
    if(n == target){
        count++;
    }
}
```

```
public static int indexAt(int[] list, int target){
    for(int i = 0; i < list.length; i++){
        if (list[i] == target){
            return i;
        }
    }
    return -1;
}
```