



Northeastern
University

Lecture 4: Fundamentals of Programming - 4

Prof. Chen-Hsiang (Jones) Yu, Ph.D.
College of Engineering

Materials are edited by Prof. Jones Yu from Prof. Charlie Wiseman's materials.

Outline

- Introduction to Computation and Programming
- Variables, I/O, Types and Strings
- Control Flow and Conditions
- Methods
- Arrays
- File I/O

Outline

- Introduction to Computation and Programming
- Variables, I/O, Types and Strings
- Control Flow and Conditions
- Methods
- **Arrays**
- File I/O

Arrays

- An *array* is a list of variables of the same type, that represents a set of related values.
- For example, say you need to keep track of the cost of 1000 items.
- You could declare 1000 double variables:
`double cost0, cost1, cost2, cost3, ...`
- Or you could use an array!

Creating Arrays

```
double[] cost = new double[1000];
```

- Start with the variable type followed by [].
- Then comes the array name (cost, x, vals, ...)
- Next is the Java keyword new followed by the type again and the size of the array in brackets.

Collection of Variables

- You can think of creating an array as declaring the same number of individual variables.
- Example declaring an array of 8 integers named `counts`:

```
int[] counts = new int[8];
```

- This is similar to (but not exactly the same as) declaring 8 separate integers:

```
int counts0, counts1, counts2, counts3, counts4, counts5, counts6, counts7;
```

Accessing Array Elements

- To actually use an individual element in the array, you **specify the *index*** of the element in brackets.
- Be careful not to confuse the two uses of brackets (**creation** vs. **use**)
- Example: creating an array of 15 integers named **values**, and setting the value at index 7 to 10:

```
int[] values = new int[15]; // create an array of 15 ints  
values[7] = 10; // assign element 7 a value of 10
```

Arrays in Memory

- Arrays are stored in memory so that all the elements in the array are next to each other, in order:

```
int[] counts = new int[8];
```

address	value	variable
1000	5	counts[0]
1004	-8	counts[1]
1008	0	counts[2]
1012	-4	counts[3]
1016	17	counts[4]
1020	4	counts[5]
1024	103	counts[6]
1028	3	counts[7]
1032		
1036		
1040		
1044		

...

Array Elements and Length

- Arrays start at index 0 and go through index size - 1
 - » Use ARRAY.length to get the size of the array
- Arrays do NOT start at index 1!
- Example of initializing an array so that all elements have an initial value of 50:

```
double[] temperatures = new double[64];  
for (int i = 0; i < temperatures.length; i++) {  
    temperatures[i] = 50;  
}
```

Out of Bounds Errors

- You always have to **ensure** that your program only uses **valid elements/indices** for an array.
- You can never access an index of less than 0, greater than or equal to the length of the array.
- If you try to access an element outside of the bounds of the array, Java will give you an **ArrayIndexOutOfBoundsException**.

```
int[] myArray = new int[10];  
myArray[0] = 5; // ok  
myArray[9] = -6; // ok  
myArray[-1] = 0; // out of bounds error!  
myArray[10] = 3; // out of bounds error!
```

Exercise

- Write a program that creates an array of 1000 integer values and initializes all 1000 values to 13.

Answer

```
int[] a = new int[1000];  
for (int i = 0; i < a.length; i++) {  
    a[i] = 13;  
}
```

Initializing Arrays

- You can also initialize arrays when you declare them using **special syntax** with **curly braces**.

- Example:

```
int[] pages = {513, 343, 279, 409, 651, 222};
```

- Above example is equivalent to:

```
int[] pages = new int[6];  
pages[0] = 513;  
pages[1] = 343;  
pages[2] = 279;  
pages[3] = 409;  
pages[4] = 651;  
pages[5] = 222;
```

Array Elements

- You can use **any element of an array** anywhere you use a variable of the same type.
 - » Assigning values
 - » In equations
 - » With input and output statements
 - » As method arguments
 - » ...

Example

```
public class ClassExamples {
    public static void main(String[] args) {
        int x;
        int[] vals = new int[5];
        for (int i = 0; i < vals.length; i++) {
            vals[i] = i*i;
        }

        x = vals[4] * vals[3] + vals[1];
        vals[0] = x - vals[2];
        vals[2] = doSomething(vals[1], vals[3]);
        for (int i = 0; i < vals.length; i++) {
            System.out.println("vals[" + i + "]= " + vals[i]);
        }
    }

    public static int doSomething(int a, int b) {
        return a * 10 + b;
    }
}
```

Arrays as Method Arguments

- Entire arrays can be passed as methods arguments.
- Array parameters in a method are a bit different than other parameters.
 - » Use **TYPE [] NAME** to indicate the parameter is an array parameter, for example: **int [] a**

Arrays as Method Arguments

- Important difference: **any changes made to array elements** in the method are **permanent** after the method is finished.
 - » In other words, changes made to the array in the method are actually being made to the array in **main()** (or whoever called the method).
 - » It actually passes **a reference** into the method.

Exercise

- Write a method named `addOne()` that increments every value in an array by one. The array must be passed as an argument to `addOne()`.

Simple Answer

```
import java.util.Scanner;

public class ClassExamples {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int[] myArray = {1,2,3,4,5};
        addOne(myArray);
    }
    public static void addOne(int[] a) {
        for (int i = 0; i < a.length; i++) {
            a[i]++;
        }
    }
}
```

Alternative Answer

```
import java.util.Scanner;

public class ClassExamples {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int[] myArray = new int[6];
        fillArray(input, myArray);
        addOne(myArray);
        printArray(myArray);
    }

    public static void fillArray(Scanner s, int[] a) {
        System.out.print("Enter " + a.length + " integers: ");
        for (int i = 0; i < a.length; i++) {
            a[i] = s.nextInt();
        }
    }

    public static void addOne(int[] a) {
        for (int i = 0; i < a.length; i++) {
            a[i]++;
        }
    }

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

Searching an Array

- Sometimes you want to search an array for a particular value or *target*.
- Look through every element and return the index of one matching element (usually the first).
- If **no element matches** the target, then we usually **return -1**, since that is never a valid index.

Exercise

- Create an integer array that contains following values:
4, 11, -3, 0, 46, 11, 9, -77, 3, 11
- Ask the user to input a number for search.
- Write a method, named `searchArray`, that takes the user's input and the array, and return back if the input exists in the array.
- Return: `the index of the found element` or `-1`

Answer

```
import java.util.Scanner;
public class ClassExamples {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int[] values = {4, 11, -3, 0, 46, 11, 9, -77, 3, 11};
        int target_value, index;

        System.out.print("Enter a value to search for: ");
        target_value = input.nextInt();
        index = searchArray(values, target_value);
        if (index == -1) {
            System.out.println("Target not found!");
        } else {
            System.out.println("Target found at index " + index);
        }
    }
    public static int searchArray(int[] haystack, int needle) {
        for (int i = 0; i < haystack.length; i++) {
            if (haystack[i] == needle) {
                return i;
            }
        }
        return -1;
    }
}
```

Take Home Points

- **Arrays** are useful when you need to **keep track of many related values**.
- Arrays are almost always used together with **loops**.
- Array elements can be used anywhere a single variable of the same type can be used.
- **Entire arrays** can be passed to methods as **array arguments**.
 - » Changes made to the array in the method affect the array in the calling method