Lab 7
Using Arrays, an Introduction

New Concepts

Grading: This lab requires the use of the grading sheet for responses that must be checked by your instructor (marked as Question) AND the submission of some programs to WebCAT (marked as Lab). Be sure to have a copy of the grading sheet, acquire the appropriate signatures and submit the sheet to your instructor in addition to submitting the appropriate labs to WebCAT.

Understanding how to manage large quantities of data is a critical requirement of designing software, and the most confusing part of it is the variety of choices available. Of these choices, arrays represent the most fundamental means of organizing, and are commonly available in most all programming languages.

An array is just a table of values, where the entire table is referenced by a single name. The problem inherent in this is that once we start using one name to refer to a number of different values, we need an additional tool for accessing the individual values. In regards to the representation of these arrays, do not be concerned as to the orientation (horizontal or vertical) as it is irrelevant.

Array x:

|       |       |       |       |       |

Each element is referenced by using the array name followed by the position in brackets. Also note that the positions start with 0. Here x has 7 elements, positions 0..6.

Array x:
Declaring an array:

To allocate an array like this in java, assuming integer values are stored in the array, use

```java
int [] x = new int [7];
```

Technical Section

Initializing / Placing data into an array:

There are a number of ways to define values for an array. One of the simplest (and LEAST useful) is at the time of declaration:

```java
int [] x = { 3, 45, 17, 2, -1, 44, 9};
```

Array `x`:

```
3   45   17   2   -1   44   9
```

The most likely place where you will use this approach is for initialization when testing a program, because it provides a convenient way of defining a set of test values. We will use this approach here for that same reason.

Most often, one changes the values one element at a time, similar to the following:

```java
x[5] = -1;
```

Array `x`:

```
3   45   17   2   -1   -1   9
```

In addition the use can be more diversified to include the arrays in more complicated ways. For example:

```java
x[2] = x[3];
```

```
3   45   17   2   -1   -1   9
```
Array $x$:

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>45</th>
<th>2</th>
<th>2</th>
<th>-1</th>
<th>-1</th>
<th>9</th>
</tr>
</thead>
</table>

See if you can determine how the state of the array changes when executing:

$$x[1] = x[x[0]];$$
Answer:

Array x:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>9</td>
</tr>
</tbody>
</table>


If this is confusing you, ask a student or your instructor for some help before going on.

Now that you have a basic understanding of how arrays and indexing works, consider the following which illustrates a more typical way that arrays will be used in your programs.

Frequently there is a need to access all of the values, to do something like set the values to zero, or find the average of the values in the array. Start with the simplest, setting all elements to zero. Once the array has been declared, one might think that the elements could be set to zero by

```
x = 0;
```

but this is not allowed. One must write a loop to repeatedly apply this same operation:

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

In truth, the arrays will be initialized to the value appropriate to the type (int, double, etc) without any explicit effort on your part. In contrast, primitive variables are not always initialized. This can be very confusing and is contrary to what happens in most programming languages, so it is suggested that arrays should be initialized by assignment as above. This part of the lab will investigate how variables and arrays are initialized.

Load the sample java application arrays.java into a BlueJ project.

Open the arrays.java source file and compile it.

Now set a breakpoint in the sampleMethod method on the line that prints “Leaving!”, and run the program.
The debugger shows some of the variables and not others. Double-click the line “int y[] =...” in the debug window which will reveal the contents of the array y.

Answer the following questions:

• Why is x missing?

• How did “a” get the values that are stored in it?

• What is different about the way “a” is initialized when compared to the way “b” is initialized?

NOTE: This question is focusing on how arrays are initialized in contrast with the way that primitives (like a) are initialized. You want to be careful about assumptions that you make concerning how java initializes variables because it is not consistent. Bottom line is that you should make sure that you initialize variables yourself. Other programming languages may not initialize at all, so developing a habit of initializing memory with your own code will be more universally correct.

Show your instructor your responses and the values in the array “y” as revealed by the debugger.
Loops for manipulating an array:

Note that there is a convenient property named **length** available with an array, and it is used in the loop below to print the values in the array. Frequently one uses loops to process array elements (illustrated here by printing them), and there are really three basic ways of “iterating” over the elements:

1. while loops
2. for loops
3. foreach loops

In this part of the lab, you will investigate these loop strategies. First, consider the simple loop to print the elements in an array.

**while version:**

```java
i = 0;
while (i<x.length)
    { System.out.println(x[i]);
        i++;
    }
```

**for version:**

```java
for (i=0; i<x.length; i++)
    { System.out.println(x[i]); }
```

**foreach version:**

```java
for (double e : x)
    { System.out.println(e); }
```

You can think of the foreach loop as a shorthand for a for loop. For example the example above is equivalent to the following code:

```java
double e = 0;
for(int i=0; i < x.length; i++){
    e = x[i];
```
System.out.println(e);

In the first two examples, the programmer is responsible to implement a counter ("i" in this example) to indicate location of successive values. The first two loop types are basically identical and only differ in where the operations are placed in the syntax.

They also represent a much more general and powerful way of iterating; i.e. they can do things a “foreach” loop can not. Nonetheless, a **foreach** loop provides a simple and effective way to iterate over the elements.

**Limitations of foreach:**

Be aware that one can only use foreach under the following restrictions:

- only to access, not change the elements
  - x[i]=3;
- only to move forward,
  - can not access in reverse
- only access every element
  - not every other element (odd positions etc)
- can not be used to do something like compare two arrays
  - x[i]>y[i]
Write a new java application named array2 which includes a simple main routine to use the following array

```java
int [] x = {3,45,17,2,-1,44,9,23,67,2,-6,-23,-100,12,5,1212};
```

and as space separated lists...

- **output the values in the array using a **for** loop**
- **output the values in the array using **foreach**
- **output the values in reverse order using a **while** loop**
- **output the values in the odd number positions (1,3,...) using a **for** loop.**
- **output the average of the values using **foreach**

*Your output should be the following:*

```
3 45 17 2 -1 44 9 23 67 2 -6 -23 -100 12 5 1212
3 45 17 2 -1 44 9 23 67 2 -6 -23 -100 12 5 1212
1212 5 12 -100 -23 -6 2 67 23 9 44 -1 2 17 45 3
45 2 44 23 2 -23 12 1212
```

*Show your code and the output to your instructor.*
Writing methods with arrays:

When using arrays as parameters, one should be mindful that when you pass arrays, changes to the formal parameters WILL change the actual parameters. This is not true of primitives (i.e. double and int). Consider the following example where the typical approach to swap two values looks like:

```java
temp = a;
a = b;
b = temp;
```

Create a new class, `ArrayExamples` and define a method called `swapInts` which is to take two integer parameters with a return type of void. Test this method from the main method. To show when, and where, the values are swapped you are to print out the values (a and b) three times:

1. In the calling method (main) before calling swapInts
2. In Swap Ints after the values are swapped
3. In the calling method (main) after the call to swapInts

Now write a second method `public void swapIntArrays(int [] a, int [] b)`. This method is to swap ALL the values in the two array parameters. Using a for loop, this method must iterate over the elements of the arrays, swapping the corresponding $i^{th}$ elements as you go through the elements. In the loop body, use code similar to the `swapInts` described above but replace “a and b” with “a[i] and b[i]”. The temp variable is not an array, it is still of type int.

Submit the class ArrayExamples including the method

`public static void swapIntArrays(int [] a, int [] b)`

to WebCAT for grading and verification that you have written the method correctly.

---

In the main method, create two arrays with values initialized in the declaration. Now test the method. If the method is working correctly the values originally in the first array should now be in the second array and vice versa. Print out the values in both arrays two times (as space separated lists like Question #2):

1. Before calling the method
2. After calling the method

You SHOULD see the values swapped in the two arrays. If WebCAT graded the submission as successful, the only question is whether or not you are using it in the main routine as directed.

Show your instructor the code and explain the results of swapInts and swapIntArray

Programming Section

Continue defining the following methods in the ArrayExamples class...

Write a method void replaceLessThan(int[] a, int b) that accepts an integer array and another integer. Iterate over the array and replace all numbers less than the number with the value of the second parameter.

If passed the array

```
int[] x = {3,45,17,2,-1,44,9,23,67,2,-6,-23,-100,12,5,1212};
```

and 12, x would be altered to contain

```
{12,45,17,12,44,12,23,67,12,12,12,12,12,12,12,1212}
```

In main, test this method. Print the array before and after calling the method.

Submit the ArrayExamples class with the

public static void replaceLessThan(int[] a, int b)

method to WebCAT for grading.
Write a second version which is very similar but returns an array with the results, leaving the original array unaltered.

```
int [] copyAndReplaceLessThan(int []a, int b)
```

As before, test this method in main. Print both the original array and the returned array which the method, copyAndReplaceLessThan, returns. Since this method returns a value (an integer array), you need to “catch” the returned value in a local variable (of type `int []`) in the calling method (main).

**For example:** Assuming you pass the array “x” to the method and use array “y” to catch the returned value. Print x before you call the method and then print both x and y after calling the method.

This is typical of the kinds of design choices that you will make, choosing whether to send the results back to the caller by modifying the input (the former) or returning a modified copy (the latter).

Submit the `ArrayExamples` class with the

```
public static int [] copyAndReplaceLessThan(int []a, int b)
```

method to WebCAT for grading.
Question 1. Answer the following questions:

• Why is x missing?

• How did “a” get the values that are stored in it?

• What is confusing about the way “a: is initialized when compared to the way “b” is initialized?