

Unit 6: Arrays

For Each Loop + Array Algorithms

Adapted from:

- 1) Building Java Programs: A Back to Basics Approach
by Stuart Reges and Marty Stepp
- 2) Runestone CSAwesome Curriculum

The Enhanced For Loop

If you are working with arrays(or other collections data structures), you can use an alternative syntax for a **for** loop (enhanced form of for loop) to iterate through items of arrays/collections.

It is also referred as **for-each loop** because the loop iterates through each element of array/collection.

```
for(data_type item : collection) {  
    ...  
}
```

The Enhanced For Loop

```
int[] numbers = {1, 3, 5, -2};  
for(int item : numbers) {  
    System.out.print(item + " ");  
}
```

Output:

1 3 5 -2

Note that there are no references to indices. Compare this with a regular for loop.

```
int[] numbers = {1, 3, 5, -2};  
for(int i = 0; i < numbers.length; i++) {  
    System.out.print(numbers[i] + " ");  
}
```

The Enhanced For Loop

```
String[] names = {"Mike", "Jesse", "Mia"};
for(String item : names) {
    System.out.print(item + " ");
}
```

Output:

Mike Jesse Mia

```
Point[] pts = {new Point(1,2), new Point(4, 5),
               new Point(2,-4)};
for(Point item : pts) {
    System.out.print(item + " ");
}
(1, 2) (4, 5) (-2, 4) // assumes toString() implemented
```

Common Mistake

```
int[] numbers = {1, 3, 5, -2};  
for(int item : numbers) {  
    item = 0;  
}  
System.out.println(Arrays.toString(numbers));
```

Output:

```
{1 3 5 -2}
```

Common Array Algorithms

Common Algorithms:

- 1) Find largest/smallest value of an array.
- 2) Compute sum, average, mode.
- 3) Determine if at least one element (or all elements) satisfy a certain property (e.g., all elements are even)
- 4) Determine the absence/presence of duplicate elements.
- 5) Determine the number of elements meeting a specific criteria. (e.g. number of positive elements)
- 6) Shift or rotate elements left or right.
- 7) Reverse the order of the elements.

We will cover some of these in lectures and the rest in labs/problem sets.

Array parameter (declare)

Arrays can be a parameter of a method.

```
public static type methodName(type[] name) {  
...}
```

Given an array, return the **largest** value of the array.

```
public static int largest(int[] array){  
    int largest = array[0];  
    for(int i = 1; i < array.length; i++){  
        if(array[i] > largest)  
            largest = array[i];  
    }  
    return largest;  
}
```

Array parameter (call)

Call a method by sending an array to its parameter.

methodName (**arrayName**) ;

```
public class MyProgram {  
    public static void main(String[] args) {  
  
        int[] iq = {126, 84, 149, 167, 95};  
        int largest = largest(iq);  
  
        System.out.println("Largest IQ = " + largest);  
    }  
    ...  
}
```

– Notice that you don't write the `[]` when passing the array.

Index of Largest Value

Given an array, return the **index** of the **largest** value of the array.

```
public static int largestIndex(int[] array) {  
    int index = 0;  
    for(int i = 1; i < array.length; i++) {  
        if(array[i] > array[index])  
            index = i;  
    }  
    return index;  
}
```

Note: This algorithm(and its variants) almost always show up on the AP free response section. Please know and memorize it!

Average

Given an array, return the **average** of the array.

```
public static double average(int[] array) {  
    int sum = 0;  
    for(int i = 0; i < array.length; i++) {  
        sum += array[i];  
    }  
    return (double)sum / array.length;  
}
```

All Even

Given an array, return whether all of the elements are even.

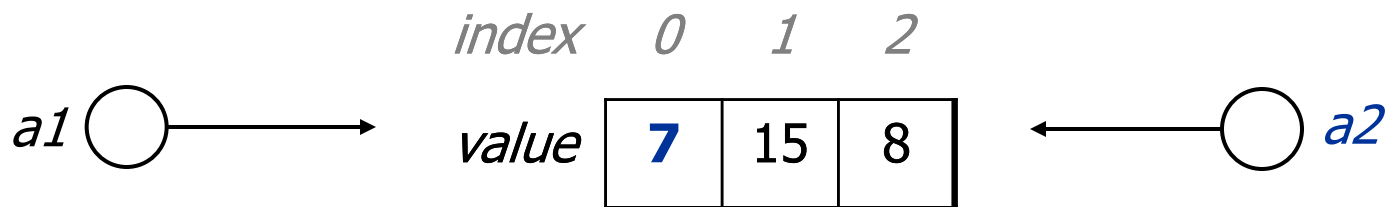
```
public static boolean allEven(int[] array) {  
    for(int i = 0; i < array.length; i++) {  
        if(array[i] % 2 != 0)  
            return false;  
    }  
    return true;  
}
```

Note: You cannot return true until all elements are checked. However, you can **early return as soon you see an odd number.**

Reference semantics (objects)

- **reference semantics:** Behavior where variables actually store the address of an object in memory.
 - When one variable is assigned to another, the object is *not* copied; both variables refer to the *same object*.
 - Modifying the value of one variable *will* affect others.

```
int[] a1 = {4, 15, 8};  
int[] a2 = a1;           // refer to same array as a1  
a2[0] = 7;  
System.out.println(Arrays.toString(a1)); // [7, 15, 8]
```



Reference Semantics

Example:

```
public static void triple(int[] numbers) {  
    for (int i = 0; i < numbers.length; i++) {  
        numbers[i] = numbers[i] * 3;  
    }  
}  
  
public static void main(String[] args) {  
    int[] arr = {0,1,2,3};  
    triple(arr);  
    System.out.println(Arrays.toString(arr));  
    // {0,3,6,9}  
}
```

Array return (declare)

A method can return an array.

```
public static type[] methodName(parameters) {
```

Example:

```
// Returns a new array with two copies of each value.  
// Example: [1, 4, 0, 7] -> [1, 1, 4, 4, 0, 0, 7, 7]  
public static int[] stutter(int[] numbers) {  
    int[] result = new int[2 * numbers.length];  
    for (int i = 0; i < numbers.length; i++) {  
        result[2 * i]      = numbers[i];  
        result[2 * i + 1] = numbers[i];  
    }  
    return result;  
}
```

Array return (call)

Storing an array returned by a method.

type[] **name** = **methodName**(**parameters**);

- Example:

```
public class MyProgram {  
    public static void main(String[] args) {  
        int[] iq = {126, 84, 149, 167, 95};  
        int[] stuttered = stutter(iq);  
        System.out.println(Arrays.toString(stuttered));  
    }  
    ...  
}
```

- Output:

[126, 126, 84, 84, 149, 149, 167, 167, 95, 95]

Array reversal question

- Write code that reverses the elements of an array.
 - For example, if the array initially stores:
`[11, 42, -5, 27, 0, 89]`
 - Then after your reversal code, it should store:
`[89, 0, 27, -5, 42, 11]`
- The code should work for an array of any size.

Does it work?

Does this work?

```
int[] numbers = [11, 42, -5, 27, 0, 89];  
// reverse the array  
for (int i = 0; i < numbers.length; i++) {  
    numbers[i] = numbers[numbers.length - 1 - i];  
  
}  
System.out.println(Arrays.toString(numbers));  
  
// [89, 0, 27, 27, 0, 89]  
// half of array was overwritten!
```

Algorithm idea

- Swap pairs of elements from the edges; work inwards:

| | | | | | | |
|--------------|----|---|----|----|----|----|
| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| <i>value</i> | 89 | 0 | 27 | -5 | 42 | 11 |
| | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |

Note: The animation above only works in powerpoint format. It will not work if you are reading this from a pdf version. The animation simply swaps 89 and 11, then swaps 0 and 42, then 27 and -5.

Flawed algorithm

- What's wrong with this code?

```
int[] numbers = [11, 42, -5, 27, 0, 89];
```

```
// reverse the array
```

```
for (int i = 0; i < numbers.length; i++) {  
    int temp = numbers[i];  
    numbers[i] = numbers[numbers.length - 1 - i];  
    numbers[numbers.length - 1 - i] = temp;  
}
```

Flawed algorithm

What's wrong with this code?

```
int[] numbers = [11, 42, -5, 27, 0, 89];  
// reverse the array  
for (int i = 0; i < numbers.length; i++) {  
    int temp = numbers[i];  
    numbers[i] = numbers[numbers.length - 1 - i];  
    numbers[numbers.length - 1 - i] = temp;  
}
```

The loop goes too far and un-reverses the array! Fixed version:

```
for (int i = 0; i < numbers.length / 2; i++) {  
    int temp = numbers[i];  
    numbers[i] = numbers[numbers.length - 1 - i];  
    numbers[numbers.length - 1 - i] = temp;  
}
```

Array reverse question

Turn your array reversal code into a `reverse` method.

- Accept the array of integers to reverse as a parameter.

```
int[] numbers = {11, 42, -5, 27, 0, 89};  
reverse(numbers);  
System.out.println(Arrays.toString(numbers));  
// {89, 0, 27, -5, 42, 11}
```

Solution: Note that this works because of reference semantics!

```
public static void reverse(int[] x) {  
    for (int i = 0; i < x.length / 2; i++) {  
        int temp = x[i];  
        x[i] = x[x.length - 1 - i];  
        x[x.length - 1 - i] = temp;  
    }  
}
```

A multi-counter problem

- Problem: Write a method `mostFrequentDigit` that returns the digit value that occurs most frequently in a number.

- Example: The number 669260267 contains:
one 0, two 2s, four 6s, one 7, and one 9.

`mostFrequentDigit(669260267)` returns 6.

- If there is a tie, return the digit with the lower value.

`mostFrequentDigit(57135203)` returns 3.

A multi-counter problem

- We could declare 10 counter variables ...

```
int counter0, counter1, counter2, counter3, counter4,  
    counter5, counter6, counter7, counter8, counter9;
```

- But a better solution is to use an array of size 10.
 - The element at index i will store the counter for digit value i .
 - Example for 669260267:

| | | | | | | | | | | |
|--------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| <i>index</i> | <i>0</i> | <i>1</i> | <i>2</i> | <i>3</i> | <i>4</i> | <i>5</i> | <i>6</i> | <i>7</i> | <i>8</i> | <i>9</i> |
| <i>value</i> | 1 | 0 | 2 | 0 | 0 | 0 | 4 | 1 | 0 | 0 |

- How do we build such an array? And how does it help?

Creating an array of tallies

```
// assume n = 669260267
int[] counts = new int[10];
while (n > 0) {
    // pluck off a digit and add to proper counter
    int digit = n % 10;
    counts[digit]++;
    n = n / 10;
}
```

| <i>index</i> | <i>0</i> | <i>1</i> | <i>2</i> | <i>3</i> | <i>4</i> | <i>5</i> | <i>6</i> | <i>7</i> | <i>8</i> | <i>9</i> |
|--------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| <i>value</i> | 1 | 0 | 2 | 0 | 0 | 0 | 4 | 1 | 0 | 0 |

Tally solution

```
// Returns the digit value that occurs most frequently in n.  
// Breaks ties by choosing the smaller value.  
public static int mostFrequentDigit(int n) {  
    int[] counts = new int[10];  
    while (n > 0) {  
        int digit = n % 10;  // pluck off a digit and tally it  
        counts[digit]++;  
        n = n / 10;  
    }  
  
    // find the most frequently occurring digit  
    int bestIndex = 0;  
    for (int i = 1; i < counts.length; i++) {  
        if (counts[i] > counts[bestIndex]) {  
            bestIndex = i;  
        }  
    }  
    return bestIndex;  
}
```

Lab 1

1) Write a method `atLeastOneOdd` that accepts an array of integers and return whether there is at least one odd number in the array.

2) Write a method `shiftRight` that accepts an array of integers and shifts each element one position to its right. The last element is wrapped back to the first.

```
int[] a1 = {11, 34, 5, 17, 56};

shiftRight(a1);
System.out.println(Arrays.toString(a1));
// {56, 11, 34, 5, 17}
```

3) Similarly, write `shiftLeft`. Test your methods!

Lab 1

4) Write the method `mode` that accepts an array of test grades(0 – 100) and return the mode, the grade that occurs most frequently. If there are multiple modes, return the smallest.

Use the multi-counter tally solution discussed in the last few slides of this lecture. What's the length of the array of tallies?