Lecture 10: Arrays II

Building Java Programs: A Back to Basics Approach
by Stuart Reges and Marty Stepp

Largest Value

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;
}
```

Index of Largest Value

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

```
public 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!

Reference semantics

Swapping values

```
public static void main(String[] args) {
    int a = 7;
    int b = 35;

    // swap a with b?
    a = b;
    b = a;

    System.out.println(a + " " + b);
}
```

- What is wrong with this code? What is its output?

- The red code should be replaced with:

```
    int temp = a;
    a = b;
    b = temp;
```

A swap method?

- Does the following `swap` method work? Why or why not?

```
public static void main(String[] args) {
    int a = 7;
    int b = 35;

    // swap a with b?
    swap(a, b);

    System.out.println(a + " " + b);
}

public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Value semantics

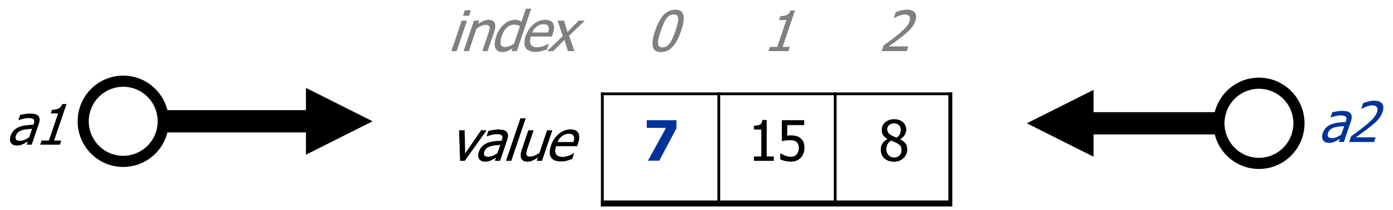
- **value semantics**: Behavior where values are copied when assigned, passed as parameters, or returned.
  - All primitive types in Java use value semantics.
  - When one variable is assigned to another, its value is copied.
  - Modifying the value of one variable does not affect others.

```
int x = 5;
int y = x;      // x = 5, y = 5
y = 17;         // x = 5, y = 17
x = 8;          // x = 8, y = 17
```

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]
```



Objects as parameters

- Arrays and objects(except String) use reference semantics. Why?
  - *efficiency.* Copying large objects slows down a program.
  - *sharing.* It's useful to share an object's data among methods.

- When an object is passed as a parameter, the object is *not* copied. The parameter refers to the same object.
  - If the parameter is modified, it *will* affect the original object.

- Arrays are passed as parameters by *reference.*
  - Changes made in the method are also seen by the caller.

Value Semantics

Example:

```
   public static void triple(int number) {
       number = number * 3;
  }
   public static void main(String[] args) {

         int x = 2;
         triple(x);
         System.out.println(x); // 2


}
```

Value Semantics

String uses value semantics like primitive types.

Example:

```
   public static void double(String str) {
       str = str + str;
  }
   public static void main(String[] args) {

           String str = “hi”;
           double(str);
           System.out.println(str); // “hi”

}
```

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}
}
```

Arrays as parameters

Array parameter (declare)

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

- Example:

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

  – You don't specify the array's length (but you can examine it).

Array parameter (call)

```
methodName(arrayName);
```

- Example:

```
public class MyProgram {
    public static void main(String[] args) {
        // figure out the average TA IQ
        int[] iq = {126, 84, 149, 167, 95};
        double avg = average(iq);
        System.out.println("Average IQ = " + avg);
    }
    ...
```

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

Array return (declare)

```
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)

```
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?

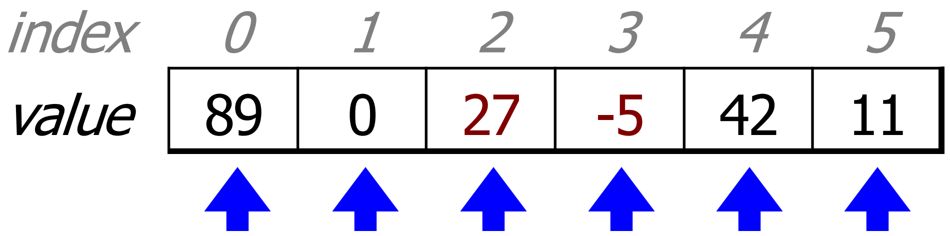
```
int[] numbers = [11, 42, -5, 27, 0, 89];
// reverse the array
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = numbers[numbers.length-1-i];

    }
System.out.println(Arrays.toString(numbers));

// [89, 0, 27, 27, 0, 89]
```

Algorithm idea

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



index	*0*	*1*	*2*	*3*	*4*	*5*
value	89	0	27	-5	42	11

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:

```
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;
    }
}
```

The `Arrays` class

- Class `Arrays` in package `java.util` has useful static methods for manipulating arrays:

Method name	Description
`binarySearch(`**array**`, `**value**`)`	returns the index of the given value in a *sorted* array (or < 0 if not found)
`equals(`**array1**`, `**array2**`)`	returns `true` if the two arrays contain same elements in the same order
`sort(`**array**`)`	arranges the elements into sorted order
`toString(`**array**`)`	returns a string representing the array, such as `"[10, 30, -25, 17]"`

- Syntax: `Arrays.`**methodName**`(`**parameters**`)`
  - `Must import java.util.*;` **`(above class header)`**

Arrays Class

```
int[] e = {0, 2, 4, 6, 8};
int index1 = Arrays.binarySearch(e,4); // index1=2
int index2 = Arrays.binarySearch(e,3); // index2<0

int[] f = {2, 6, 8, 0, 4};
boolean a = Arrays.equals(e,f); //a=false

Arrays.sort(f); //f is now {0,2,4,6,8}
boolean a = Arrays.equals(e,f); // true
a = e.equals(f);
//above is false
a = e == f;
// above is false
```

Lab 1

- Write a method `computeOdd` that accepts an array of integers and returns the sum of all odd values in the array.

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

```
int a2 = computeOdd(a1); //a2=11+5+17=33
```

Lab 1

- Write a method `swapAll` that accepts two arrays of integers as parameters and swaps their entire contents. You have to use a loop to swap individual elements.
  - Assume that the two arrays are the same length.

```
int[] a1 = {12, 34, 56};
int[] a2 = {20, 50, 80};
swapAll(a1, a2);
System.out.println(Arrays.toString(a1));   // [20, 50, 80]
System.out.println(Arrays.toString(a2));   // [12, 34, 56]
```

Lab 1

- Write a method `merge` that accepts two arrays of integers and returns a new array containing all elements of the first array followed by all elements of the second.

```
int[] a1 = {12, 34, 56};
int[] a2 = {7, 8, 9, 10};

int[] a3 = merge(a1, a2);
System.out.println(Arrays.toString(a3));
// [12, 34, 56, 7, 8, 9, 10]
```

Lab 2

Watch the video Keyboard Inputs and Mouse Inputs on my channel before completing labs 2 and 3.

Write a program that displays a rectangle on the screen. The rectangle moves left/right/up or down by 10 pixels if the user presses the corresponding arrow keys. The program also displays a circle on the screen. The circle follows the mouse.

If the user clicked on the mouse, the rectangle jumps to the mouse's position.

Implement both the void keyPressed() and void mousePressed() methods.

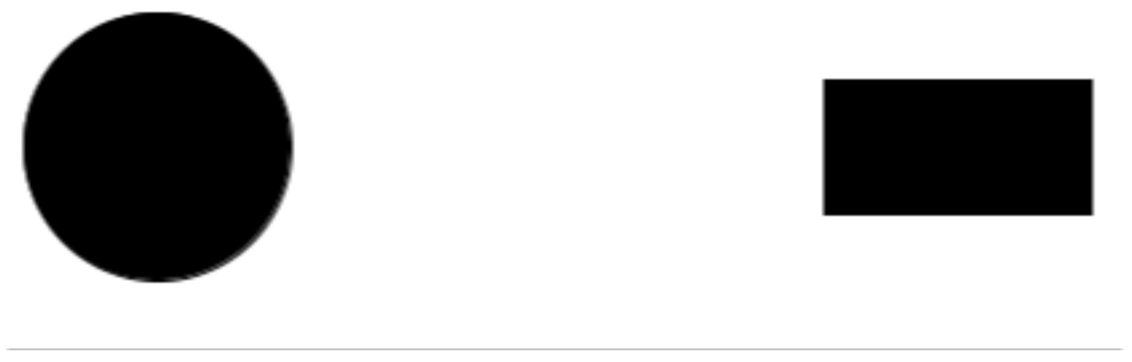
Lab 3: Mouse Collision I

Write a program that that display a circle whose fill is red if the mouse is inside the circle and blue if it is outside.

Use the built in dist(x1,y1,x2,y2) to compute the distance between (x1,y1) and (x2,y2).

Lab 3: Mouse Collision II

Modify the previous program to display a circle and a rectangle that is initially filled with the color black. If the mouse is inside of the circle, the circle turns red. If it is inside of the rectangle, the rectangle turns blue. If the mouse is not in either shape, both remains black.



Lab 3: Mouse Collision II