Arrays

Ye Yang
Stevens Institute of Technology

Introduction

- Array is a useful and powerful aggregate data structure presence in modern programming languages
- Arrays allow us to store arbitrary sized sequences of primitive values or sequences of references to objects
- Arrays allow easy access and manipulation to the values/objects that they store
- Arrays are indexed by a sequence of integers
- Classes can use arrays as instance variables to store databases of value/references

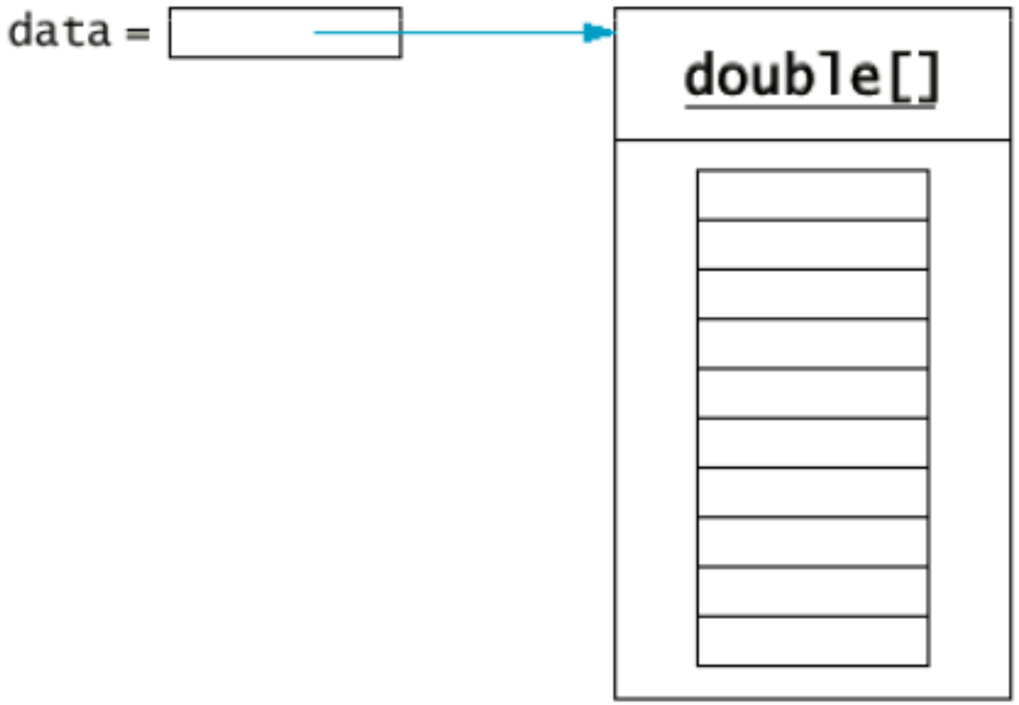
Array

- new is used to construct a new array:

```
new double[10]
```

- Store 10 double type variables in an array of doubles

```
double[] data = new double[10];
```



Arrays

```java
int[] scores = new int[5];
```

- Declares an array of size 5
- First item starts at index 0
- Arrays are initialized by default in Java
- This prints five zeros

```java
int[] scores   = new int[5];
for (int i=0; i<5; i++) {
  System.out.println(scores[i]);
};
```

Arrays

- We can also initialize the elements with our own values

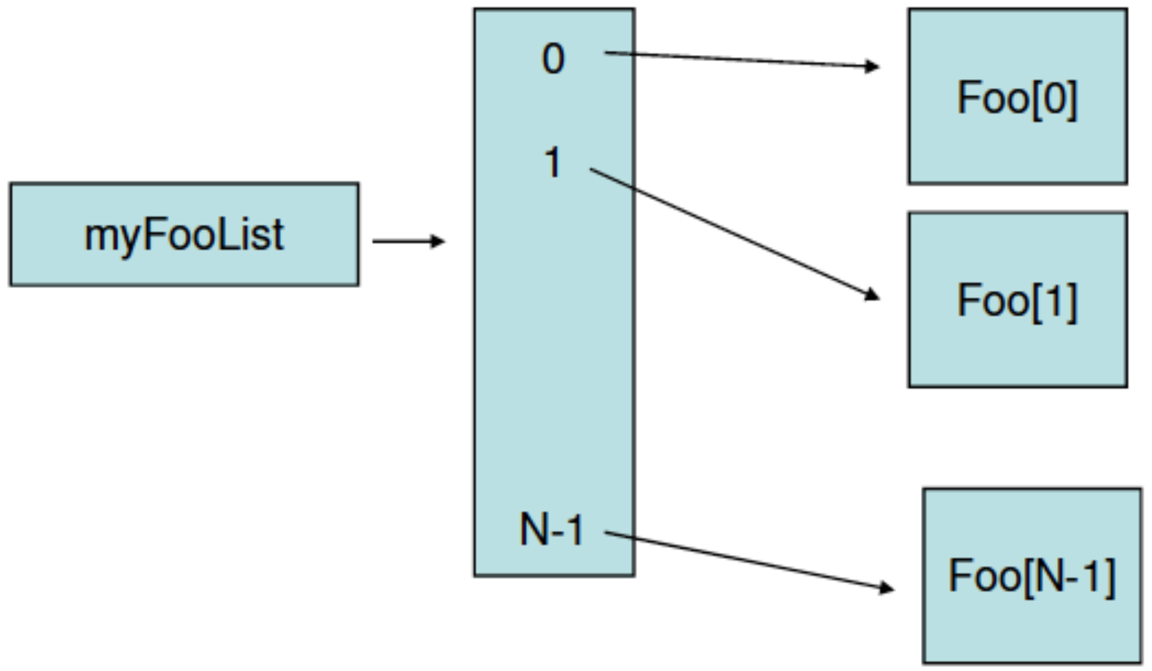
```java
String[] names = {"Sally", "Jill", "Hal", "Rick"};
System.out.println(names.length);
// length above is data field, not a method
```

- The elements of an array can also have user defined types

```java
Person[] people;
int n       = 3+4;
people      = new Person[n];
people[0] = new Person("Elliot","Koffman","123"
    ,1942);
```

Array of Object References

```
class foo() { ….}
foo[ ]  myFooList = new foo[N];
```



Arrays

- There is an enhanced for loop for collections, arrays included
- Rather than

```
for (int i=0; i<5; i++) {
  System.out.println(scores[i]);
};
```

- We can write

```
for (int i : scores) {
  System.out.println(scores[i]);
};
```

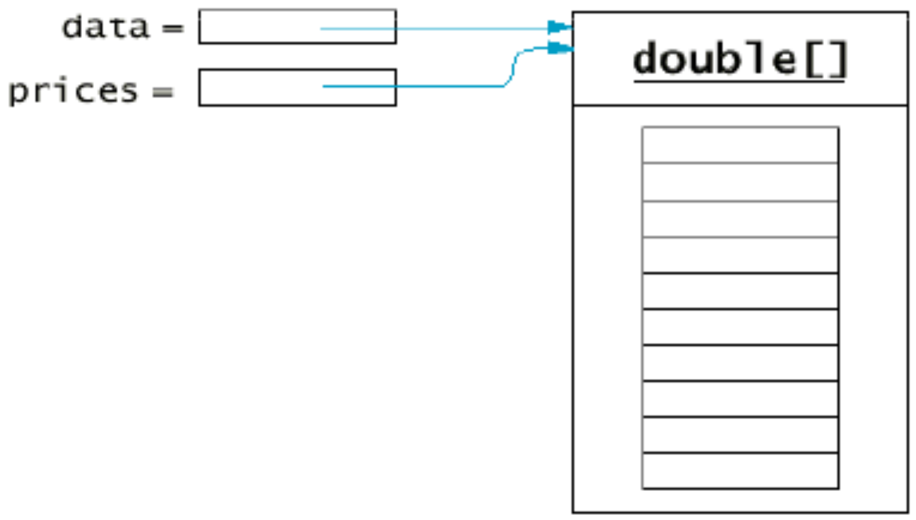
Arrays

- Arrays have fixed length
- Array is a homogeneous data structure:
  - each of its members stores the same type (either primitive or reference)
  - Operator [ ] is used to access array elements
    - data[4] = 29.95;
- Use length attribute to get array length.
  - data.length. (Not a method!)
    - Length: a **public final int** instance variable
- Array indices go from 0 to one less than the length of the array

Copying Arrays

- Copying an array reference yields a second reference to the same array

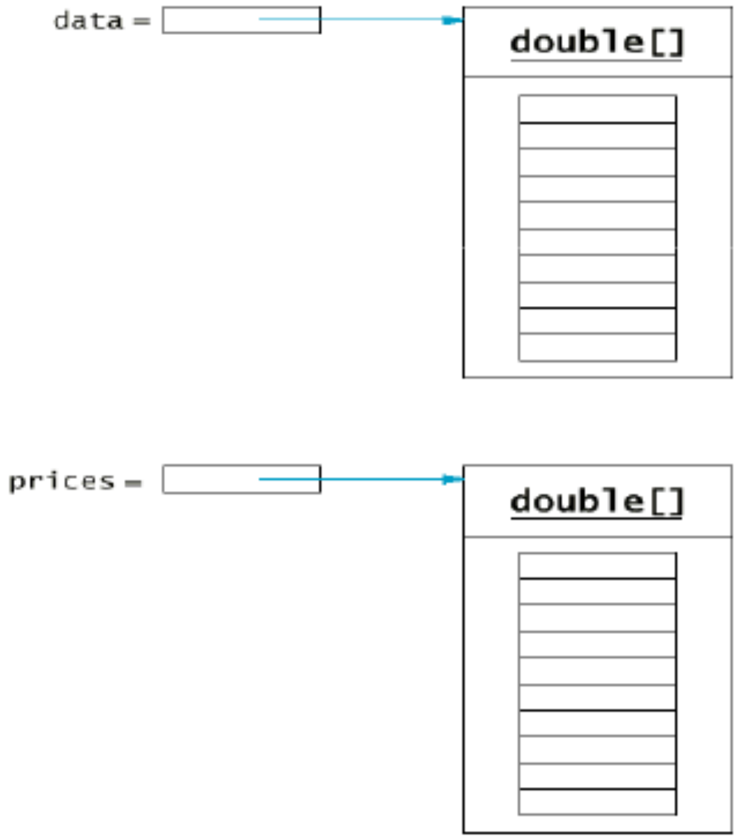
```
double[] data = new double[10];
  // fill array . . .
  double[] prices = data;
```



Cloning Arrays

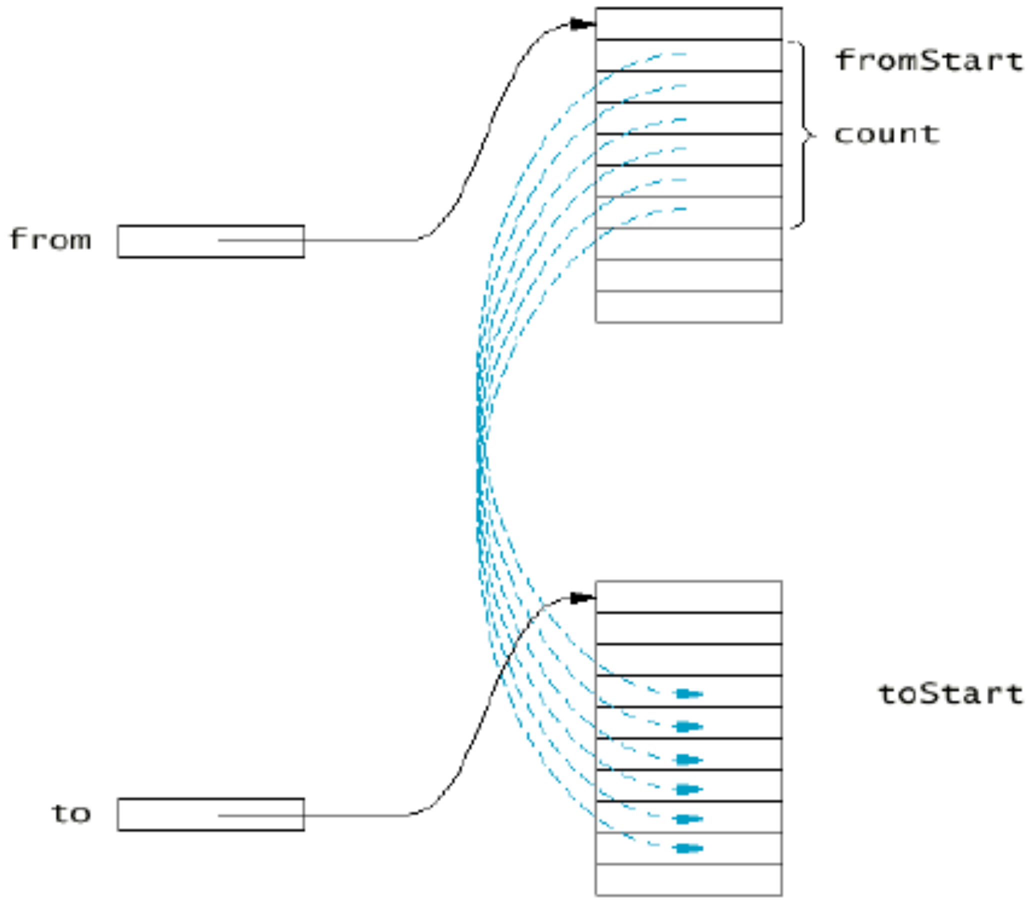
- Use clone to make true copy

```
double[] prices = (double[])data.clone();
```



Copying Array Elements

System.arraycopy(from, fromStart, to, toStart, count);



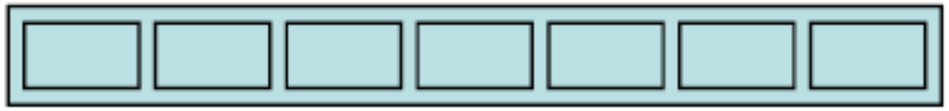
The java.lang.System.arraycopy() method copies a source array from a specific beginning position to the destination array from the mentioned position.

No. of arguments to be copied are decided by **count** argument.

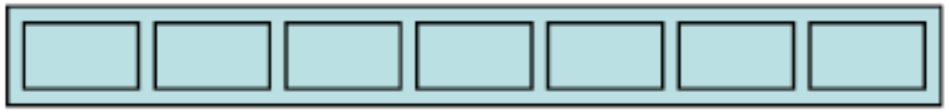
The components at **fromStart** to **fromStart + count– 1** are copied to destination array from **toStart** to **toStart + count – 1**

Shifting Elements

- Shift all elements to Right by 1 starting at index i



- Shift all elements left by 1 starting at index i (i>0)



Swapping Array Elements

- Suppose you want to swap two elements in the array, say entries with indices i and j.
- Assuming we are dealing with an array of ints
  - int temp = A[i]; // save a copy of A[i] in temp
  - A [i] = A[j]; // copy the content of A[j] to A[i]
  - A[j] = temp; // copy the content of temp to A[j]
- Note that : A[i]= A[j] and A[j] = A[i] do not swap content
- Exercise: Reverse an array using swaps

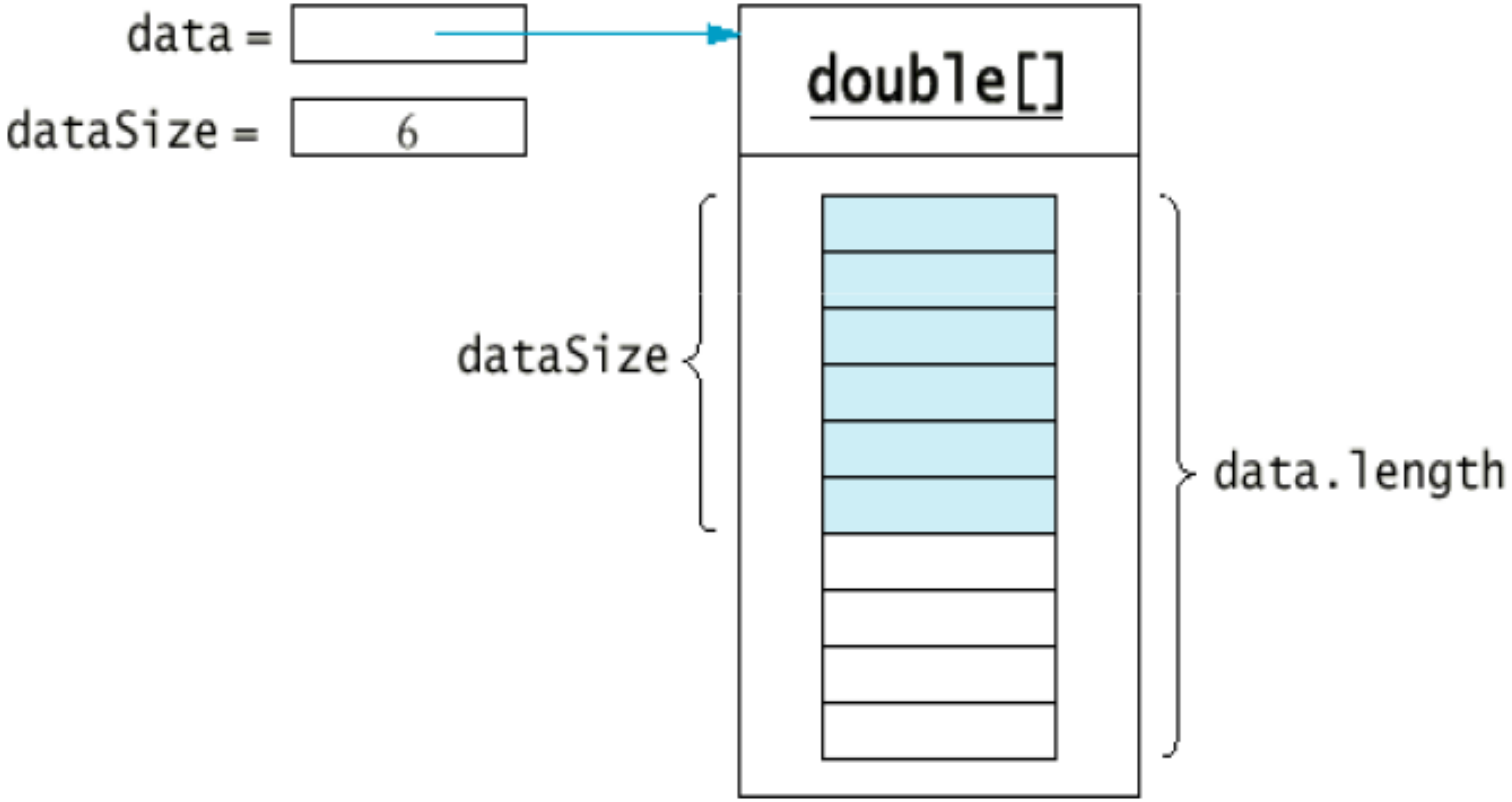
Accessing Arrays

- int[] a = new int[]{4, 2, 0, 1, 3};
- system.out.println(a[0]);
- if (a[5] == 0) ...some statement
- if the value computed for the index is less than 0, or greater than OR EQUAL TO the length of the array
  - trying to access the member at an illegal index causes Java to throw the ArrayIndexOutOfBoundsException which contains a message showing what index was attempted to be accessed

Partially Filled Arrays

- Array.length = maximum capacity of the array
- Usually, array is partially filled
- Need companion variable to keep track of current size
  - final int capacity = 100;
  - double[] data = new double[capacity];
  - int size = 0;
- Update size as array is filled:
  - data[size] = x;
  - size++;

Partially Filled Arrays



Partially Filled Arrays

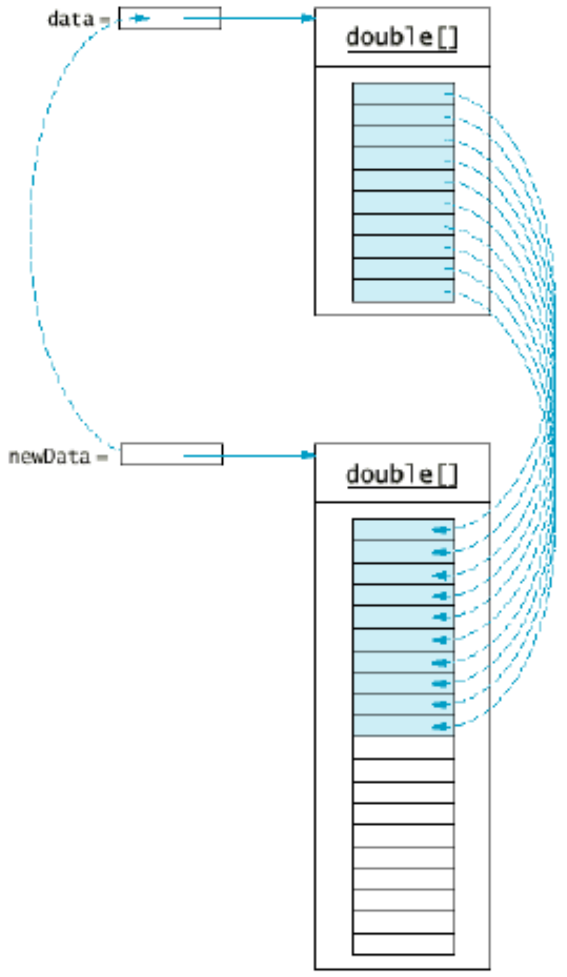
- Remember to stop at dataSize-1 when looking atarray elements:

```
for (int i = 0; i < dataSize; i++)
        sum = sum + data[i];
```

- Be careful not to overfill the array

```
if (dataSize >= data.length)
        System.out.println("Sorry--array  full");
```

Resizing an Array



Dynamic Arrays

- Arrays are typically static structures
- However we can design a new array classthat is dynamic (that is, you never run outof space)
- Java already has a dynamic array classcalled ArrayList
- See Java API for ArrayList class
  - https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

ArrayLists (Next Lecture)

java.util

Class ArrayList<E>

```
java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractList<E>
            java.util.ArrayList<E>
```

Parameter Passing is Call-by-Value

- In Java all arguments are call-by-value
  - If the argument is a primitive type, its value, not its address, are passed to the method
  - The method cannot modify the argument value and have this modification remain after returning
  - If the argument is of class type, it can be modified using its own methods and the changes are permanent
- Other languages also support call-by-reference

Parameter Passing is Call-by-Value

```
public void foo(Dog d) {
    d = new Dog("Snoopy"); // creates the "Snoopy" dog
}

Dog aDog = new Dog("Pluto"); // creates the "Pluto" dog
// aDog points to the "Pluto" dog
foo(aDog);
// aDog still points to the "Pluto" dog
```

Arrays as parameter/return (declare)

- Arrays can be passed as parameters and returned from methods.

```
public static type name(type[] name) { // pass array parameter
public static type[] name(parameters) { // return array
```

- This method takes an array of doubles, and returns a new array of rounded ints:

```
public static int[] roundAll(double[] realNums) {
    int[] roundedNums = new int[realNums.length];
    for (int i = 0; i < realNums.length; i++) {
        roundedNums[i] = (int) Math.round(realNums[i]);
    }
    return roundedNums;
}
```

Arrays as parameter/return (call)

- Below is an example usage of the roundAll method from the previous slide:

```
import java.util.*; // to use Arrays public class
MyProgram {
    public static void main(String[] args) {
        double[] realNumbers = {5.5, 7.31, 8.09, -3.234234, 2.0, 0.0};
        int[] roundedNumbers = roundAll(realNumbers);
        System.out.println(Arrays.toString(roundedNumbers));
    }
    ...
}
// Output: [5, 7, 8, -3, 2, 0]
```

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:

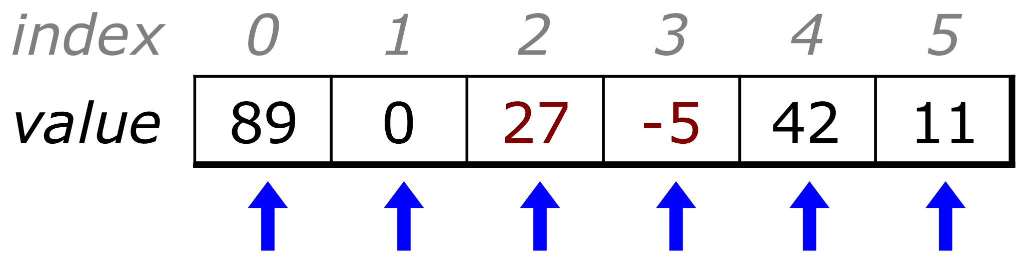
```
    int temp = a;
    a = b;
    b = temp;
```

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.
    - Hint: think about swapping various elements...

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;
}
```

- 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 2

- 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);
```

  - How do we write methods that accept arrays as parameters?
  - Will we need to return the new array contents after reversal?

    …

Array parameter (declare)

```
public static <type> <method>(<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>[] <method>(<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>*** `=` ***<method>***`(`***<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]
```

Multidimensional Arrays

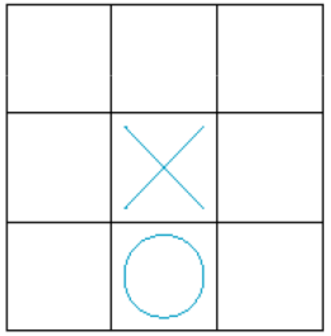
- Some application solutions require tables with multiple dimensions
  - Modeling a matrix require a 2-dimensional array or table
  - Modeling an application that require 3-dimensional array
- Example: in Graphics, representing a point (x, y, z)

Two Dimensional Arrays

```
final int ROWS = 3;
final int COLS = 3;
double[][] matrix = new double[ROWS][COLS];

for (int i =0; i<ROWS; i++) {
    for (int j=0; j<COLS; j++) {
        System.out.println(matrix[i][j]);
    }
}
```

- Example: Tic Tac Toe board



char[][] board = new char[3][3];
board[i][j] = 'x';

Memory Allocation

- Java (and many other language compilers) allocate memory for 2D arrays as an array of 1D arrays