

## Lectures 6.1 - 6.2

# Arrays

## Outline

---



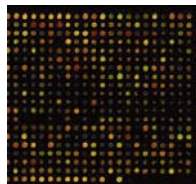
- Why arrays?
- Array declaration
- Array processing examples
- Multi-dimensional arrays
- Command line arguments
- Array-based classes and algorithms
  - Polynomial
  - Sorting
  - Merging

## Why arrays?

- So far, our programs contained objects like `grade` and `turtle`.
- We will now see how objects like `grade[100]` and `turtle[20]` can be created and managed
- Such objects are called arrays
- Arrays lend themselves to situations in which you have to store and process a series of objects of the same type.



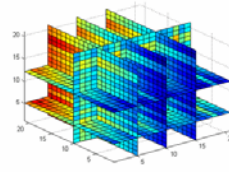
1-dimensional array



2-dimensional array



3-dimensional array



n-dimensional array  
(impossible to draw)

## Array representation: examples

**DNA file:** `GTTCAATGATTTTAACCACACTTATAGGTAGATGA ... TGATTC` (1000 bases)

This data can be viewed as an indexed sequence of bases:

	0	1	2	3	4	...	994	995	996	997	998	999
<b>DNA array:</b>	G	T	T	C	A	...	T	G	A	T	T	C

- Which base appears in location 995 ?
- Where is the last location of base A ?
- Which pattern appears in locations 510 to 523 ?
- How many times each base appears in the sequence ?
- Does the pattern CGT appear in the sequence ?
- Does the pattern C?G appear (where ? is any one base) ?
- Does the pattern C\*G appear (where \* is any sequence of 0 or more bases) ?
- Given two sequences  $s_1$  and  $s_2$ , compute the function  $\text{similarity}(s_1, s_2) = \% \text{ of locations in which they have identical bases}$
- And so on.

## Array representation: examples

DNA array:

0	1	2	3	4	...	994	995	996	997	998	999
G	T	T	C	A	...	T	G	A	T	T	C

```
Char[] dna = new char[1000];
// get the contents of dna from a given dnaFile (omitted)

// Where base appears in location 995 ?
base = dna[995];

// Which is the last location of base A ?
found = -1;
for (i = 0, i <= 1000, i++)
    if (dna[i] == 'A')
        found = i;

// Which pattern appears in locations 510 to 523 ?
String pattern = "";
for (i = 510, i <= 523, i++) pattern = pattern + dna[i];
```

## Array representation: examples

Array = indexed collection of objects

Each object is characterized, and referred to, by a numeric index

An array of size  $n$  is indexed from 0 to  $n-1$

Example:

	0	1	2	3	4	5	6	7	8	9
grades	55	78	90	45	71	100	23	85	81	64

```
int[] grades = new int[10];
// get the grades from a file, or interactively from the user (omitted)
// Some sample array operations:
int x = grades[4];
grades[6] = x;
System.out.println(grades[6]);           // will print 71
grades[3] = grade[3] + 5;
System.out.println(grades[3]);           // will print 50
System.out.println(grades[2] * 1.1);     // will print 99
```

- If array  $x$  is of type `int`, then any one of its elements can be treated as an `int` variable and can play any role that an `int` variable plays in Java
- Same rule holds for any other array type.

## Array declaration

Syntax:

```
dataType[] arrayName = new dataType [ arrayLength ]
```

Examples

```
char[] dna = new char[1000];  
int[] downloads = new int[100];  
boolean[] flags;  
flags = new boolean[10];
```

Arrays that contain  
primitive data types

```
String[] words = new String[500];  
Turtle[] turtles = new Turtle[10];  
WindMill[] windMills = new WindMill[100];
```

Arrays that contain  
references to objects

```
// move the third turtle 50 steps forward:  
turtles[2].moveForward(50);  
  
// Compute the total gain of the wind mill farm  
for (i = 0, i <= 100, i++)  
    yield = yield + windMill[i].getYield();
```

## Outline

- Why arrays?
- Array declaration
- ➔ ■ Array processing examples
- Multi-dimensional arrays
- Command line arguments
- Array-based classes and algorithms
  - Polynomial
  - Sorting
  - Merging

## Array processing example 1

```
import java.util.Scanner;
public class ArrayDemo1 {
    public static void main(String[] args){

        // Declares an array of a user-defined size
        int[] elements;
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter number of elements: ");
        int size = scan.nextInt();
        elements = new int[size];

        // Prompts the user to enter the array elements
        for (int j = 0 ; j < size ; j++) {
            System.out.print("enter element " + j + ": ");
            elements[j] = scan.nextInt();
        }

        // Prints the array elements in reverse order
        for (int j = size - 1 ; j >= 0 ; j--)
            System.out.println(elements[j]);
    }
}
```

Prompts the user for a series of values, then prints them backwards

```
ca C:\WINDOWS\system32\cmd.exe
D:\demo>java ArrayDemo1
Enter number of elements: 7
enter element 0: 10
enter element 1: 20
enter element 2: 15
enter element 3: 7
enter element 4: 90
enter element 5: 2
enter element 6: 103
103
2
90
7
15
20
10
D:\demo>
```

## Array processing example 2

```
import java.util.Scanner;
public class DigitsCount {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter a number: ");
        int number = scan.nextInt();
        int[] digitCounter = new int[10];

        // For each digit in the number, increment its counter
        while (number>0) {
            int digit = (int)(number%10);
            digitCounter[digit]++;
            number/=10;
        }
        // Print the histogram
        for (int i = 0; i < 10; i++) {
            System.out.print(i + " ");
            for (int j = 0; j < digitCounter[i]; j++)
                System.out.print("*");
            System.out.println();
        }
    }
}
```

Count how many times each digit appears in a given number

```
ca C:\WINDOWS\system32\cmd.exe
D:\demo>javac DigitsCount.java
D:\demo>java DigitsCount
Enter a number:
2276326212903200
0 ****
1 *
2 ****
3 **
4
5
6 **
7 *
8
9 *
```

## Array length

- In Java, arrays are implemented as objects
- Each array object has a field called `length`, containing the array size

Instead of writing:

```
int[] grades = new int[100];
...
for (int i = 0; i < 100; i++) {
    System.out.print(grades[i]);
}
```

It's much better to write:

```
final static int size = 100;
...
int[] grades = new int[size];
...
for (int i = 0; i < grades.length; i++) {
    System.out.print(grades[i]);
}
```

## Initializer lists

Examples:

```
int[] firstTenPrimes = {3, 5, 7, 11, 13, 17, 19, 23, 29, 31};
char[] vowels = {'a', 'o', 'i', 'u', 'e'};
```

The rules of the game:

- An initializer list is an array
- When defining an initializer list, there is no need to use `new`
- The array length is determined from the size of the list
- An initializer list can be defined only as part of the array declaration statement.

## Array of characters vs. String

- A String is not an array of characters
- Strings are immutable; Array entries can be changed
- A String can be constructed from an array of characters:

```
char[] c = new char[20];
// Populate c with values (omitted)
String s = new String(c);
```

### A natural correspondence:

- `c.length` VS. `s.length()`
- `c[i]` VS. `s.charAt(i)`

## Array processing example 3

```
import java.util.Scanner;
public class FindVowels {
    static char[] vowels = {'a', 'e', 'i', 'o', 'u'};

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter a word: ");
        String word = scan.nextLine();
        System.out.println("The word has " +
            (containsVowel(word) ? "" : "no ") +
            "vowel(s)");
    }

    public static boolean containsVowel(String s) {
        for (int j = 0; j < s.length(); j++)
            for (int k = 0; k < vowels.length; k++)
                if (s.charAt(j) == vowels[k])
                    return true;
        return false;
    }
}
```

Checks if a given string  
has one or more  
vowels in it.

```
C:\WINDOWS\system32\cmd.exe
```

```
D:\demo>java FindVowels
Enter a word: baby
The word has vowel(s)

D:\demo>java FindVowels
Enter a word: gypsy
The word has no vowel(s)
```

## Arrays as parameters

```
public class ContainsZero {  
  
    public static void main(String[] args) {  
        int[] x = new int[10];  
        int[] y = new int[5000];  
        for (int j = 0 ; j < x.length ; j++) x[j]=j;  
        for (int j = 0 ; j < y.length ; j++) y[j]=j+1;  
        System.out.println(containsZero(x));  
        System.out.println(containsZero(y));  
    }  
  
    public static boolean containsZero(int[] a) {  
        boolean result = false;  
        for (int j = 0 ; j < a.length ; j++)  
            if (a[j] == 0)  
                result = true;  
        return result;  
    }  
}
```

Checks if a given array of integers contains a zero.

```
cmd C:\WINDOWS\system32\cmd.exe
```

```
D:\demo>java ContainsZero  
true  
false
```

## Arrays are objects

```
int[] x = new int[10];  
for (int j = 0 ; j < x.length ; j++) {  
    x[j] = j;  
}  
  
int[] z = x;  
z[3] = 1000;  
System.out.print(x[3]);  
// Will print 1000.
```

Memory	
...	...
10102	45119 x
10103	45119 z
...	...
45118	10
45119	0
45120	1
45121	2
45122	1000
45123	4
45124	5
45125	6
45126	7
45127	8
45128	9
...	...

} array data

## Array of objects

**Arrays of primitive types:** the elements of an array can be values of primitive types like `int`, `char`, etc. Example:

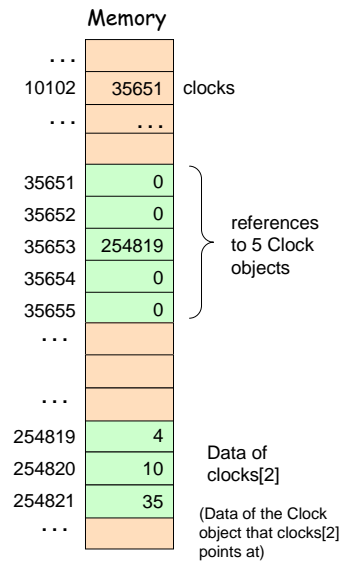
```
int[] grades = new int[100];
```

**Arrays of objects:** the elements of an array can also be references to objects. Example:

```
Clock[] clocks = new Clock[5];
```

This statement reserves space for 5 references to `Clock` objects that may or may not be constructed later :

```
Clocks[2] = new Clock(4, 10, 35);
// ...
int minute = Clocks[2].getMinute();
```

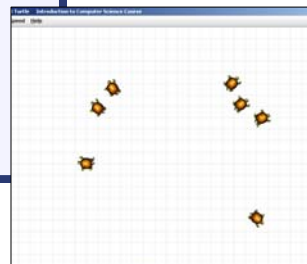


## Array of objects: example

```
public class TurtleJungle {
    static final int STEP = 20;
    static final int n_TURTLES = 7;
    public static void main(String[] args) {
        Turtle[] turtles = new Turtle[n_TURTLES];
        for (int i = 0; i < turtles.length; i++) {
            turtles[i] = new Turtle();
            turtles[i].turnLeft((int)(Math.random()*360));
            turtles[i].tailUp();
        }

        while(true)
            for (int i = 0; i < turtles.length; i++)
                turtles[i].moveForward(STEP);
    }
}
```

Create some turtles and send them to move in random directions



## Outline

- Why arrays?
- Array declaration
- Array processing examples
- ➔ ■ Multi-dimensional arrays
- Command line arguments
- Array-based classes and algorithms
  - Polynomial
  - Sorting
  - Merging

## Multidimensional data

### Math conventions:

v: ( 3 -19 7 13 8 )  
 $v_2$  holds the value -19

m:  $\begin{pmatrix} 5 & 2 & 7 & 3 & 8 & 5 \\ 2 & 4 & 8 & 12 & 7 & -3 \\ 3 & 19 & 5 & 8 & 5 & 1 \\ -2 & 3 & 6 & 4 & 9 & 5 \end{pmatrix}$   
 $m_{2,4}$  holds the value 12

### Java conventions:

v: ( 3 -19 7 13 8 )  
 $v[2]$  holds the value 7

m:  $\begin{pmatrix} ( 5 & 2 & 7 & 3 & 8 & 5 ) \\ ( 2 & 4 & 4 & 12 & 7 & -3 ) \\ ( 3 & 19 & 5 & 8 & 5 & 1 ) \\ ( -2 & 3 & 6 & 4 & 9 & 5 ) \end{pmatrix}$   
 $m[2][4]$  holds the value 5

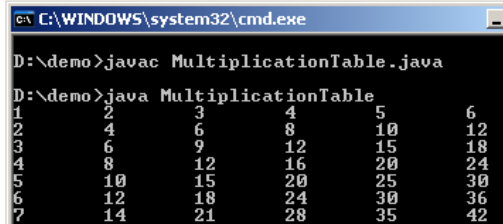
## Multidimensional array example: multiplication table

```
public class MultiplicationTable {
    public static void main(String[] args){
        final int n = 10;

        int[][] table = new int[n][n];
        for (int j = 0 ; j < table.length ; j++)
            for (int k = 0 ; k < table[0].length ; k++)
                table[j][k] = j * k;

        for (int j = 1 ; j < n ; j++){
            for (int k = 1 ; k < n ; k++)
                System.out.print(table[j][k] + "\t");
            System.out.println();
        }
    }
}
```

In Java, a multidimensional array is implemented as an array of arrays



```
C:\WINDOWS\system32\cmd.exe
D:\demo>javac MultiplicationTable.java
D:\demo>java MultiplicationTable
1 2 3 4 5 6
2 4 6 8 10 12
3 6 9 12 15 18
4 8 12 16 20 24
5 10 15 20 25 30
6 12 18 24 30 36
7 14 21 28 35 42
```

## Multidimensional array example: a diagonal matrix

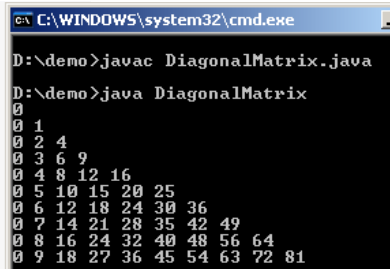
```
public class DiagonalMatrix {
    public static void main(String[] args){

        int[][] mat = new int[10][];

        for (int j=0 ; j<mat.length ; j++)
            mat[j] = new int[j+1];

        for (int j = 0 ; j < mat.length ; j++)
            for (int k = 0 ; k < mat[j].length ; k++)
                mat[j][k] = j * k;

        for (int j = 0 ; j < mat.length ; j++) {
            for (int k = 0 ; k < mat[j].length ; k++)
                System.out.print(mat[j][k] + " ");
            System.out.println();
        }
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
D:\demo>javac DiagonalMatrix.java
D:\demo>java DiagonalMatrix
0
0 1
0 2 4
0 3 6 9
0 4 8 12 16
0 5 10 15 20 25
0 6 12 18 24 30 36
0 7 14 21 28 35 42 49
0 8 16 24 32 40 48 56 64
0 9 18 27 36 45 54 63 72 81
```

- The Java implementation of a multidimensional array is an array of arrays
- Therefore, each "row" can have a different length.

## Initializing multi-dimensional arrays

A multidimensional array can be declared and initialized using an **initializer list**:

Example:

```
int[][] hadamardMatrix = {
    { 1,  1,  1,  1},
    {-1,  1, -1,  1},
    {-1, -1,  1,  1},
    {-1, -1, -1,  1}
};
```

Note that the initializer list is a list of lists:

```
{{...}, {...}, ... , {...}}
```

## Command line arguments

When the user invokes a class from the OS shell, she can pass a line of parameters to the `main` method using the syntax:

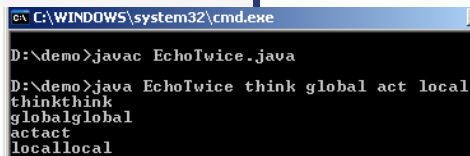
```
Java ClassName parametersLine
```

The OS shell breaks the line (following the class name) into tokens, and stores them in the entries of the `String[] args` array

The `args` array is accessible to the program's code like any other array.

Example:

```
public class EchoTwice {
    public static void main(String[] args) {
        for(int i = 0 ; i < args.length ; i++) {
            for(int j = 0; j < 2 ; j++)
                System.out.print(args[i]);
            System.out.println();
        }
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
D:\demo>javac EchoTwice.java
D:\demo>java EchoTwice think global act local
thinkthink
globalglobal
actact
locallocal
```

## Outline

---

- Why arrays?
- Array declaration
- Array processing examples
- Multi-dimensional arrays
- Command line arguments
- Array-based classes and algorithms



- Polynomial
- Sorting
- Merging

## Polynomial functions

---

A Polynomial function, aka "Polynomial", is a real-valued function of the following form:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0$$

Where  $n$  is a nonnegative integer, called the polynomial degree,  $a_0, a_1, a_2, \dots, a_n$  are real-valued coefficients, and  $x$  is the Polynomial argument, i.e. the number at which the Polynomial is evaluated.

Example:  $f(x) = x^3 - x$

Note that a Polynomial is completely characterized by its coefficients.  
For example, the last Polynomial can be characterized by  $(-1, 0, 0, 1)$

## Polynomial class: client view

The following example constructs the Polynomial  $p(x) = 4x^3 + 2x^2 - 3x + 1$   
And then computes the value  $p(2)$

```
public class PolynomialDemo1 {  
  
    public static void main(String args[]) {  
  
        // p(x) = 4x^3 + 2x^2 - 3x + 1  
  
        double[] pCoefficients = {1, -3, 2, 4};  
  
        Polynomial p = new Polynomial(pCoefficients);  
        System.out.println(p);  
        System.out.println("p(2) = " + p.valueAt(2));  
    }  
}
```

```
C:\WINDOWS\system32\cmd.exe
```

```
D:\demo>java PolynomialDemo1  
4.0x^3 + 2.0x^2 - 3.0x + 1.0  
p(2) = 35.0
```

## Polynomial class: fields and constructor

```
// A polynomial. A real valued function of the form  
// p(x) = a0 + a1*x + a2*x^2 + ... + an*x^n.  
  
public class Polynomial {  
  
    // The polynomial's coefficients are kept in an array:  
    // the coefficient of x^i is stored in coefficients[i].  
    private double[] coefficients;  
  
    // Constructs a polynomial from a given array of coefficients.  
    public Polynomial(double[] coefficients) {  
        this.coefficients = new double[coefficients.length];  
        for(int i = 0; i < coefficients.length; i++)  
            this.coefficients[i] = coefficients[i];  
    }  
  
    // Accessor: returns the coefficient of x^k  
    public double getCoefficient(int k) {  
        return k < coefficients.length ? coefficients[k] : 0;  
    }  
}
```

server

We will unfold the class implementation gradually, using several slides.

Each slide shows a subset of relevant class members.

```
double[] pCoefficients = {1, -3, 2, 4};  
Polynomial p = new Polynomial(pCoefficients);
```

client

## Displaying the Polynomial

**client**

```
double[] pCoefficients = {1, -3, 2, 4};
Polynomial p = new Polynomial(pCoefficients);
// prints the coefficients (omitted)
System.out.println("p(x) = " + p);
```

```
C:\WINDOWS\system32\cmd.exe
D:\demo>java PolynomialDemo3
Coefficients: 1.0 -3.0 2.0 4.0
p(x) = 4.0x^3 + 2.0x^2 - 3.0x + 1.0
```

**server**

```
// Converts the polynomial to a string representation
public String toString() {
    int degree = getDegree();
    if (degree == 0) return "" + coefficients[0];
    if (degree == 1) return coefficients[1] + "x + " + coefficients[0];
    String s = coefficients[degree] + "x^" + degree;
    for (int i = degree-1; i >= 0; i--) {
        if (coefficients[i] == 0) continue;
        else if (coefficients[i] > 0) s = s + " + " + (coefficients[i]);
        else if (coefficients[i] < 0) s = s + " - " + (-coefficients[i]);
        if (i == 1) s = s + "x";
        else if (i > 1) s = s + "x^" + i;
    }
    return s;
}
```

## Accessors (some of them)

```
// Returns the polynomial degree.
public int getDegree() {
    int degree = coefficients.length - 1;
    while (coefficients[degree] == 0.0) {
        degree--;
    }
    return degree;
}
```

```
// Returns the coefficients of the polynomial.
public double[] getCoefficients() {
    double[] copy = new double[coefficients.length];
    System.arraycopy(coefficients, 0, copy, 0, coefficients.length);
    return copy;
}
```

## Adding Polynomials: client view

$$f(x) = 7x^2 + 5x + 2 \quad g(x) = 8x^3 - 10 \quad f(x) + g(x) = ?$$

```
public class PolynomialDemo2 {  
  
    public static void main(String args[]) {  
  
        // p(x) = 4x^3 + 2x^2 - 3x + 1  
        double[] pCoefficients = {1, -3, 2, 4};  
        Polynomial p = new Polynomial(pCoefficients);  
        System.out.println("\n" + "p(x) = " + p);  
  
        // q(x) = x^4 + 2x - 7  
        double[] qCoefficients = {-7, 2, 0, 0, 1};  
        Polynomial q = new Polynomial(qCoefficients);  
        System.out.println("\n" + "q(x) = " + q);  
  
        Polynomial pPlusq = p.plus(q);  
        System.out.println("\n" + "(p+q)(x) = " + pPlusq);  
        System.out.println("\n" + "pPlusq(1) = " + pPlusq.valueAt(1));  
    }  
}
```

```
C:\WINDOWS\system32\cmd.exe  
D:\demo>java PolynomialDemo2  
p(x) = 4.0x^3 + 2.0x^2 - 3.0x + 1.0  
q(x) = 1.0x^4 + 2.0x - 7.0  
(p+q)(x) = 1.0x^4 + 4.0x^3 + 2.0x^2 - 1.0x - 6.0  
pPlusq(1) = 0.0
```

## Outline

- Why arrays?
- Array declaration
- Array processing examples
- Multi-dimensional arrays
- Command line arguments
- Array-based classes and algorithms
  - Polynomial
  - ➔ • Sorting
  - Merging

## Sorting

### The sorting task:

- Input: an array of values, e.g. (5, 2, 10, 8, 5)
- Output: a sorted array, e.g. (2, 5, 5, 8, 10)
- The values can be any object whose data type has an order
- Sorting is a common computational task:
  - Sort your contacts list
  - Sort pages by their pageRank
  - Sorting before searching

We will show one sorting algorithm, applied to integers

### Later in the course:

- More sorting algorithms
- Java implementations designed to sort any type data.



## Selection sort

### Pseudo code:

```
n = array.length - 1
step 0:
  for (i from 0 to n)
    maxi = index of the maximum in this range:
    switch a[maxi] with a[n]
step 1:
  for (i from 0 to n-1)
    maxi = index of the maximum in this range:
    switch a[maxi] with a[n-1]
step j:
  for (i from 0 to n-j)
    maxi = index of the maximum in this range:
    switch a[maxi] with a[n-j]
```

### Main idea:

In the  $j$ th step, find the  $j$ th largest number and put it in the  $n-j$  location of the array.

Unsorted: 

3	10	4	8	15	1	7	4
---	----	---	---	----	---	---	---

Step 0: 

3	10	4	8	4	1	7	15
---	----	---	---	---	---	---	----

Step 1: 

3	7	4	8	4	1	10	15
---	---	---	---	---	---	----	----

Step 2: 

3	7	4	1	4	8	10	15
---	---	---	---	---	---	----	----

 Etc.

Sorting animation: <http://www.sorting-algorithms.com/insertion-sort>

## Selection sort

```
public class SelectionSort {
    public static void main(String[] args) {
        int[] data = {3, 10, 4, 6, 15, 1, 7, 4};
        selectionSort(data);
        for (int j=0 ; j<data.length ; j++)
            System.out.println(data[j]);
    }

    public static void selectionSort (int[] a) {
        int n = a.length-1;
        for (int j=0 ; j<=n ; j++) {
            // find index of maximal element in a[0]...[n-j]
            int maxi = 0;
            for (int i=1 ; i<=n-j ; i++) {
                if (a[i] > a[maxi])
                    maxi = i;
            }
            // switch
            int temp=a[n-j]; a[n-j]=a[maxi]; a[maxi]=temp;
        }
    }
}
```

### "In-Place" sorting:

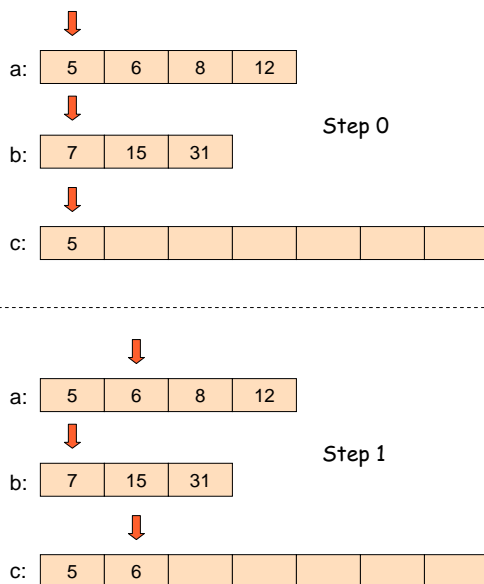
The data variable holds a reference to the array's values.

When we pass test as a parameter to the sorting method, the passing mode is "call by reference"

Therefore, the method can actually change the array values

Such methods are said to have "side effects", and must be used with great caution.

## Merging



**Input:** sorted arrays a and b

**Output:** merged and sorted array c

**Algorithm (informal):**

```
int[] c = new int[a.length + b.length]
```

```
int i = j = k = 0
```

In each step:

```
if (a[i] < b[j])
```

```
    c[k++] = a[i++]
```

```
else
```

```
    c[k++] = b[j++]
```

If one of the arrays empties before the other one, copy all the remaining values from the other one to c.