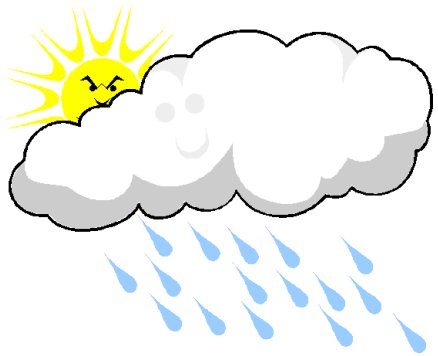
Lecture 9: Arrays

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

Can we solve this problem?

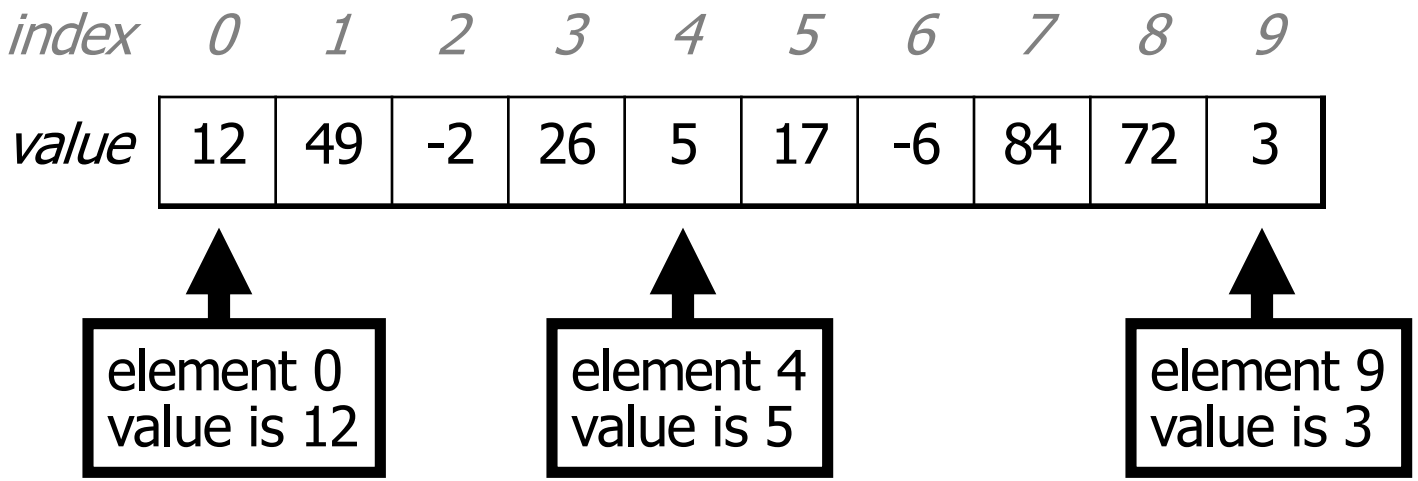
- Consider the following program (input underlined):

```
How many days' temperatures? 7
Day 1's high temp: 45
Day 2's high temp: 44
Day 3's high temp: 39
Day 4's high temp: 48
Day 5's high temp: 37
Day 6's high temp: 46
Day 7's high temp: 53
Average temp = 44.6
4 days were above average.
```



Arrays

- **array**: object that stores many values of the same type.
  - **element**: One value in an array.
  - **index**: A 0-based integer to access an element from an array.

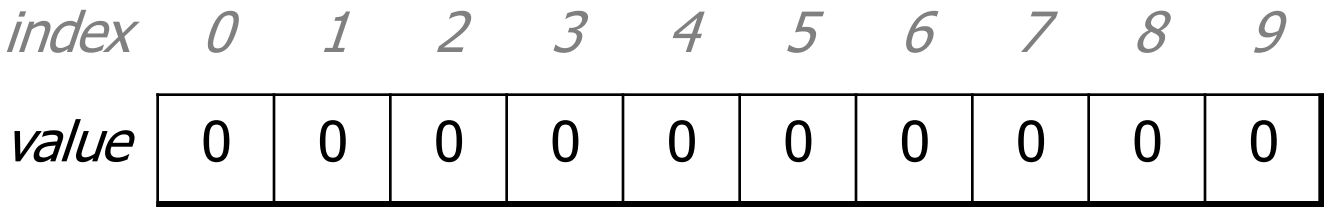


Array declaration

type`[]` **name** `=` `new` **type**`[`**length**`];`

- Example:

```
int[] numbers = new int[10];
```



Array declaration, cont.

- The length can be any integer expression.

```
int x = 2 * 3 + 1;
int[] data = new int[x % 5 + 2];
```

- Each element initially gets a "zero-equivalent" value.

Type	Default value
`int`	`0`
`double`	`0.0`
`boolean`	`false`
`String` or other object	`null` (means, "no object")

Accessing elements

```
name[index]                    // access
name[index] = value;           // modify
```

- Example:

```
numbers[0] = 27;
numbers[3] = -6;

System.out.println(numbers[0]);
if (numbers[3] < 0) {
    System.out.println("Element 3 is negative.");
}
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
value	**27**	0	0	**-6**	0	0	0	0	0	0

Arrays of other types

```
double[] results = new double[5];
results[2] = 3.4;
results[4] = -0.5;
```

index	*0*	*1*	*2*	*3*	*4*
value	0.0	0.0	**3.4**	0.0	**-0.5**

```
boolean[] tests = new boolean[6];
tests[3] = true;
```

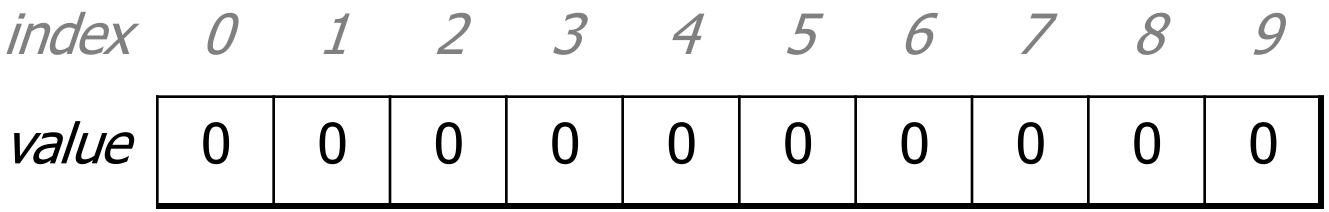
index	*0*	*1*	*2*	*3*	*4*	*5*
value	false	false	false	**true**	false	false

Out-of-bounds

- Legal indexes: between **0** and the **array's length - 1**.
  - Reading or writing any index outside this range will throw an `ArrayIndexOutOfBoundsException`.

- Example:

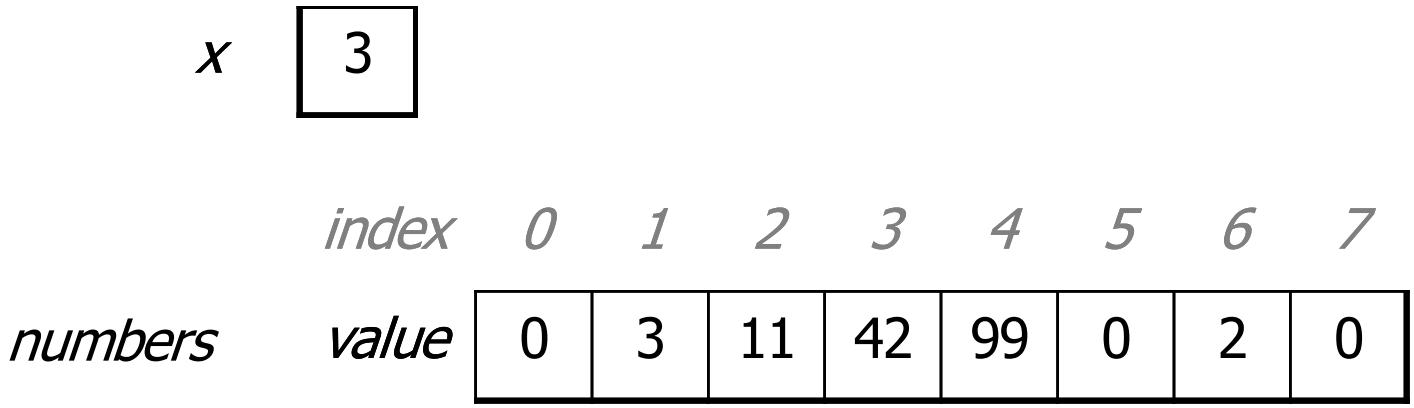
```
int[] data = new int[10];
System.out.println(data[0]);        // okay
System.out.println(data[9]);        // okay
System.out.println(data[-1]);       // exception
System.out.println(data[10]);       // exception
```



Accessing array elements

```
int[] numbers = new int[8];
numbers[1] = 3;
numbers[4] = 99;
numbers[6] = 2;

int x = numbers[1];
numbers[x] = 42;
numbers[numbers[6]] = 11; // use numbers[6] as index
```



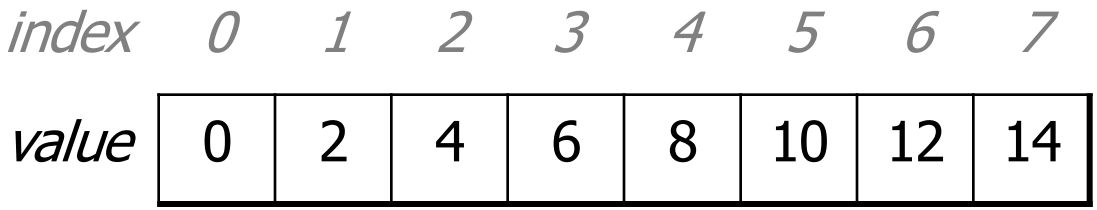
Arrays and `for` loops

- It is common to use `for` loops to access array elements.

```
for (int i = 0; i < 8; i++) {
    System.out.print(numbers[i] + " ");
}
System.out.println();  // output: 0 3 11 42 99 0 2 0
```

- Sometimes we assign each element a value in a loop.

```
for (int i = 0; i < 8; i++) {
    numbers[i] = 2 * i;
}
```



The `length` field

- An array's `length` field stores its number of elements.

  name`.length`

```
for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + " ");
}
// output: 0 2 4 6 8 10 12 14
```

  - It does not use parentheses like a String's `.length()`.

- What expressions refer to:
  - The last element of any array?
  - The middle element?

Quick array initialization

type`[]` **name** `=` `{`**value**`,` **value**`,` ... **value**`};`

– Example:

```
int[] numbers = {12, 49, -2, 26, 5, 17, -6};
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	12	49	-2	26	5	17	-6

– Useful when you know what the array's elements will be
– The compiler figures out the size by counting the values

"Array mystery" problem

- **traversal**: An examination of each element of an array.
- What element values are stored in the following array?

```
int[] a = {1, 7, 5, 6, 4, 14, 11};
for (int i = 0; i < a.length - 1; i++) {
    if (a[i] > a[i + 1]) {
        a[i + 1] = a[i + 1] * 2;
    }
}
```

"Array mystery" problem

- **traversal**: An examination of each element of an array.
- What element values are stored in the following array?

```
int[] a = {1, 7, 5, 6, 4, 14, 11};
for (int i = 0; i < a.length - 1; i++) {
    if (a[i] > a[i + 1]) {
        a[i + 1] = a[i + 1] * 2;
    }
}
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	1	7	10	12	8	14	22

Limitations of arrays

- You cannot resize an existing array:

```
int[] a = new int[4];
a.length = 10;              // error
```

- You cannot compare arrays with == or `equals`:

```
int[] a1 = {42, -7, 1, 15};
int[] a2 = {42, -7, 1, 15};
if (a1 == a2) {  ... }              // false!
if (a1.equals(a2)) {  ... }         // false!
```

- An array does not know how to print itself:

```
int[] a1 = {42, -7, 1, 15};
System.out.println(a1);             // [I@98f8c4]
```

Arrays.toString

```
public static void main(String[] args) {
      int[] a = {0, 14, 4, 6, 8};
      System.out.println(a);
}
Output: I@674f1c67
```

- Prints out the address not the contents of a.

  `Arrays.toString` accepts an array as a parameter and returns a `String` representation of its elements.

```
System.out.println("a is " + Arrays.toString(a));
```

Output:

```
a is [0, 14, 4, 6, 8]
```

float vs double

- **float**: 32-bit data type representing real numbers to about 7 decimal places.

- **double:** 64-bit data type representing real numbers to about 16 decimal places.

Since Processing is graphics intensive, it is recommended that floats are used instead of doubles. In fact, float is the default type for decimal numbers in Processing. All of the built-in math functions in Processing returns floats. The math functions from the standard Math library in Java returns doubles.

```
double a = Math.sqrt(9); // a = 3.0 but is a double
float b = sqrt(9); // b = 3.0 but is a float
```

float vs double

Processing's `Math` methods

Method name	Description
`abs(`*value*`)`	absolute value(int or float)
`ceil(`*value*`)`	rounds up(int)
`exp(`*value*`)`	exponential, base e
`log(`*value*`)`	logarithm, base e
`map(x,x1,x2,y1,y2)`	linear map x from [x1,x2] to [y1,y2]
`pow(`*base*`, `*exp*`)`	*base* to the *exp* power(float)
`random(`*value*`)`	random `float` from [0,value)
`random(`*value1*`, `*value2*`)`	random `float` from [value1,value2)
`round(`*value*`)`	nearest whole number(int)
`sqrt(`*value*`)`	square root
`sin(`*value*`)`, `cos(`*value*`)` `tan(`*value*`)`	sine/cosine/tangent of an angle in radians
`atan(value)` `atan2(dy,dx)`	inverse tangent (-PI/2, PI/2) inverse tangent (-PI, PI)
`degrees(`*value*`)` `radians(`*value*`)`	convert degrees to radians and back

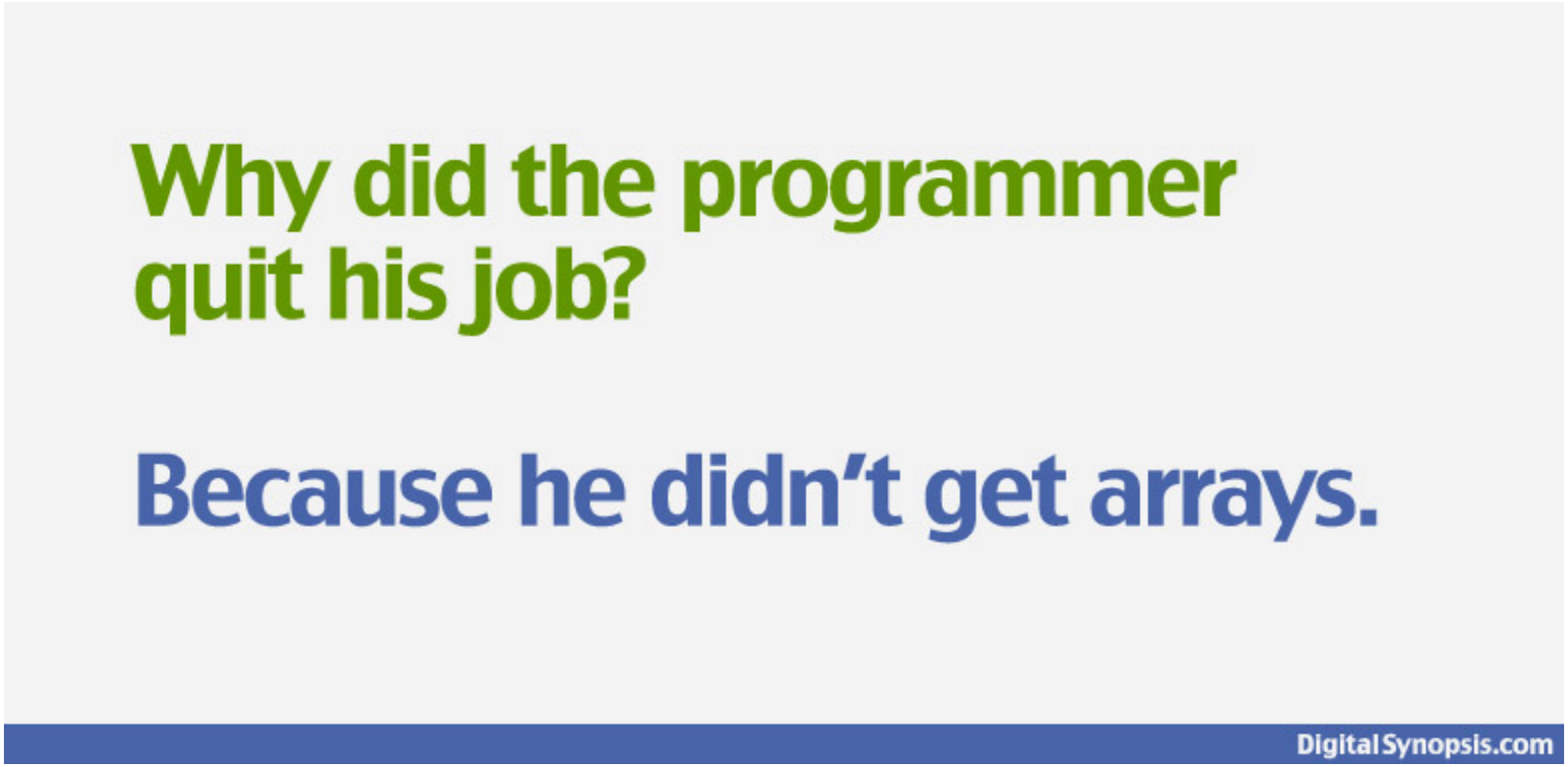
Constant	Description
`PI`	3.1415926...

Arrays

```
String[] a=[“hip”, “hip”];

//hip hip arrays!
```

Arrays



He didn't get arrays and he didn't get a raise.

Array Lab 1

Write the method `average` which accepts an int array and returns the average of the values.

Write the method `countAboveAve` which accepts an int array and returns the number of values that are above the average. You must call `average`.

Write two methods `largest and smallest` which accept an int array and returns the largest value and smallest value, respectively.

Also write the main method with an array and check to make sure your methods work!

```
public static double average(int[] array){}
public static int countAboveAve(int[] array){}
public static int largest(int[] array){}
public static int indexOfsmallest(int[] array){}
```

Array Lab 2

Write a **Processing** program generate a large number of balls with random starting position that moves left and right at random speeds.

You'll need 4 arrays: an array for x positions, an array for y positions, an array for its xspeed and one more for its diameter(or radius).

The setup() method randomizes position, speed and diameters with a for loop.

The draw() method draw and moves and bounces the balls.

Array Lab 2

Add the method `indexLargest` which returns the index of the largest ball. Draw this largest ball in blue. The .pde template file can be found on Classroom.