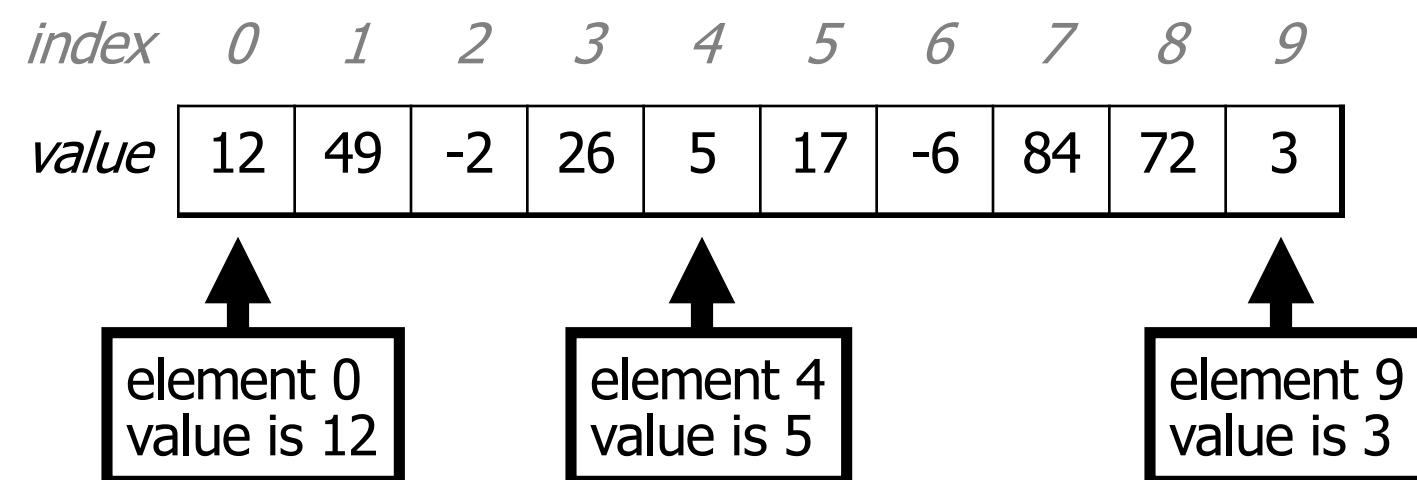


# Lecture 10: Arrays

AP Computer Science Principles

# 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] ;
```

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

# Array declaration, cont.

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

Type	Default value
int	0
float	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;
```

```
println(numbers[0]); // 27  
if (numbers[3] < 0) {  
    println("Element 3 is negative.");  
}
```

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	27	0	0	-6	0	0	0	0	0	0

# Arrays of other types

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

<i>index</i>	0	1	2	3	4
<i>value</i>	0.0	0.0	<b>3.4</b>	0.0	<b>-0.5</b>

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

<i>index</i>	0	1	2	3	4	5
<i>value</i>	false	false	false	<b>true</b>	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];
println(data[0]);           // okay
println(data[9]);           // okay
println(data[-1]);          // exception
println(data[10]);          // exception
```

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

# 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
```

$x$     3

	<i>index</i>	0	1	2	3	4	5	6	7
<i>numbers</i>	<i>value</i>	0	3	11	42	99	0	2	0

# Arrays and for loops

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

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

*index    0    1    2    3    4    5    6    7*

<i>value</i>	0	2	4	6	8	10	12	14
--------------	---	---	---	---	---	----	----	----

# 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
```

- What expressions refer to:
  - The last element of any array? `numbers[numbers.length-1]`
  - The middle element? `numbers[numbers.length/2]`

# Quick array initialization

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

- Example:

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

<i>index</i>	0	1	2	3	4	5	6
<i>value</i>	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;  
    }  
}
```

<i>index</i>	0	1	2	3	4	5	6
<i>value</i>	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!  
// the above compare whether a1 and a2 are the same objects.
```

- An array does not know how to print itself:

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

# printArray(array)

```
int[] e = {1, -2, 4};  
printArray(e);
```

Output:

```
[0] 1  
[1] -2  
[2] 4
```

# Finding the largest Value

Given an array, return the the largest value in the array.

```
void setup() {  
    int[] a = {1,-5,5,7,56,23};  
    int large = largest(a); // large = 56  
}  
  
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;  
}
```

# Reference semantics

# Value semantics

- **value semantics:** When primitive variables (`int`, `float`, `boolean`) and `String` are passed as parameters, their values are copied. Modifying the parameter will not affect the variable passed in.

```
void setup() {  
    int x = 23;  
    strange(x);  
    println("2. x = " + x);  
}  
  
void strange(int x) {  
    x = x + 1;  
    println("1. x = " + x);  
}
```

**Output:**

```
1. x = 24  
2. x = 23
```

# 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:

```
void setup() {  
    int x = 2;  
    triple(x);  
    println(x); // 2  
  
}  
void triple(int number) {  
    number = number * 3;  
}
```

# Value Semantics

**String uses value semantics like primitive types.**

**Example:**

```
void setup() {  
    String str = "hi";  
    twice(str);  
    println(str); // "hi"  
}  
void twice(String str) {  
    str = str + str;  
}
```

# Reference Semantics

Example:

```
void setup() {  
    int[] arr = {0,1,2,3};  
    triple(arr);  
    printArray(arr); // {0,3,6,9}  
}  
  
void triple(int[] numbers) {  
    for (int i = 0; i < numbers.length; i++) {  
        numbers[i] = numbers[i] * 3;  
    }  
}
```

# Arrays

```
String[] a = ["hip", "hip"];
```

```
//hip hip arrays!
```

# Arrays

Why did the programmer quit his job?

Because he didn't get arrays.

(He didn't get arrays and 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 the method `indexOfLargest` which accepts an int array and returns the index of the largest value. If there are multiple largest values, return the index of the first one.

Write the method `equals` which accept two integer arrays and determine if they have the same value.

# Lab 1 Outline

Write the setup() method to test your methods.

```
void setup() {  
    int[] a = {1, 3, 2, 5, -4, -7};  
    println(indexOfLargest(a)); // should be 3  
}  
  
float average(int[] array) {  
}  
  
int countAboveAve(int[] array) {  
}  
  
int indexOfLargest(int[] array) {  
}  
  
boolean equals(int[] a, int[] b) {  
}
```

# Lab 2

Due: Tues, Oct 6.

In Array-1, do:

makeMiddle, commonEnd, midThree and swapEnds

In Array-2, do:

countEvens, lucky13, no14 and sum28

# Array Lab 3

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

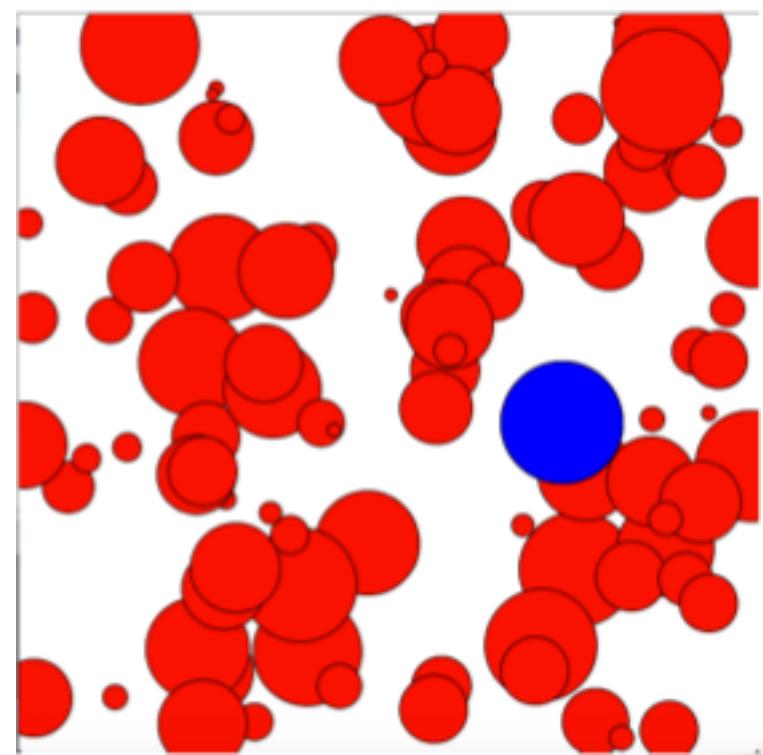
You'll need 4 arrays: an array for x positions, an array for y positions, an array for its xspeed and one more for its diameter(or radius).

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

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

# Array Lab 3

Add the method `indexLargestDiameter` which returns the index of the largest ball. Draw this largest ball in blue. It is recommended that you try to do this on your own but there is a .pde template file on my github website if you need some help.



# References

Part of this lecture is taken from the following book.

- 1) Stuart Reges and Marty Stepp. Building Java Programs: A Back to Basics Approach. Pearson Education. 2008.