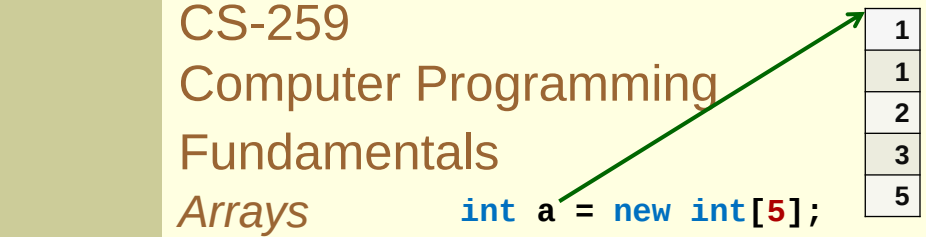
# CS-259
# Computer Programming Fundamentals
## *Arrays*




Instructor:
Joel Castellanos
e-mail:
joel@unm.edu

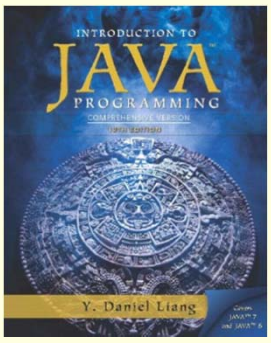
*And in such indexes, although small pricks*
*To their subsequent volumes, there is seen*
*The baby figure of the giant mas*
*Of things to come at large.*

--William Shakespeare, Troilus and Cressida

10/5/2016

## Textbook & Reading Assignment

*Introduction to java Programming (10th Edition)* by Y. Daniel Liang




Read by Wednesday: Sept 28
Chapter 7: Arrays
Sections 1 through 5.

Study Questions:
7.5, 7.6, 7.7, 7.8, 7.10

Read by Friday: Sept 30
Chapter 7: Arrays
Sections 6 through end of chapter.

2

## An Index (*plural: indexes*)

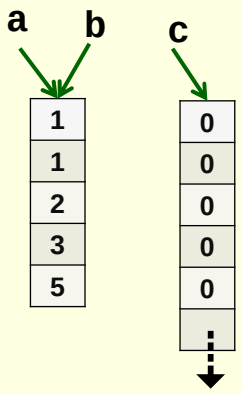
- In publishing, an **index** is a list of words or phrases, "headings", and associated pointers "locators" to where useful material relating to that heading can be found in a document.
- In a traditional back-of-the-book index, the headings will include names of people, places and events, and concepts selected by a person as being relevant and of interest to a possible reader of the book.
- The pointers are typically page numbers, paragraph numbers or section numbers.
- In a library catalog the words are authors, titles, subject headings, etc., and the pointers are call numbers.

3

## The Array

In Java, an ***array*** is collection of data items allocated in sequential memory locations that can be selected by indices computed at run-time.

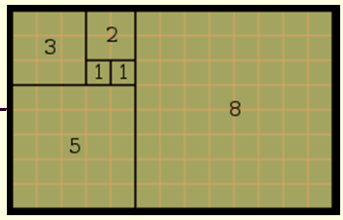
```
1) int[] a;
2) int[] b = {1, 1, 2, 3, 5};
3) int[] c = new int[10000000];
4) c[32767] = 9;
5) a = b;
6) System.out.println(a[i]);
```




No matter how large an array, a particular element can be loaded or stored in constant time.

4

## Arrays: Fibonacci Series




```
1)  //Example without an array
2)  int f0 = 1;
3)  int f1 = 1;
4)  int f2 = f1 + f0;
5)  int f3 = f2 + f1;
6)  int f4 = f3 + f2;
7)  int f5 = f4 + f3;
8)  int f6 = f5 + f4;
9)
10) System.out.println(f0 + " " + f1 + " "
11)   + f2 + " " + f3 + " " + f4 + " " + f5
12)   + " " + f6);
13)
14) //1 1 2 3 5 8 13
```

5

## Arrays: Fibonacci Series

```
1)  //Example with an array
2)  int[] f = new int[7];
3)  f[0] = 1;
4)  f[1] = 1;
5)  f[2] = f[1] + f[0];
6)  f[3] = f[2] + f[1];
7)  f[4] = f[3] + f[2];
8)  f[5] = f[4] + f[3];
9)  f[6] = f[5] + f[4];
10)
11) System.out.println(f[0] + " " + f[1]
12)   + " " + f[2] + " " + f[3] + " " + f[4]
13)   + " " + f[5] + " " + f[6]);
14) //1 1 2 3 5 8 13
```

1. `int[] f` declares `f` as a ***reference*** to an `int` array.

2. `f = new int[7]` reserves memory for 7 `int` variables and sets `f` to reference that block of memory.

6

## Array Versatility: Fibonacci Series

```
1)  //Example array
2)  int[] f = new int[14];
3)  f[0] = 1;
4)  f[1] = 1;
5)  for (int i=2; i<f.length; i++)
6)  { f[i] = f[i-1] + f[i-2];
7)  }
8)
9)  for (int i=0; i<f.length; i++)
10) { System.out.print(f[i] + " ");
11) }
12) System.out.println();
13)
14) //1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

In Java, arrays hava a ***field*** **`.length`** not a method **`.length()`.**

By contrast, a **`String`** has a ***method*** **`.length()`** not a field **`.length`.**

7

## Quiz: Arrays and Fibonacci Series

In the Java code below, which lines can be swapped (have their positions interchanged) without causing an error or changing the output?

```
1)  int[] f = new int[16];
2)  f[0] = 1;
3)  f[1] = 1;
4)  for (int i=2; i<f.length; i++)
5)  {
6)    System.out.println(f[i-2]);
7)    f[i] = f[i-1] + f[i-2];
8)  }
```

a) 1 & 2 b) 3 & 4
c) 4 & 5 d) 4 & 6 e) 6 & 7

8

## Helper Method: `printArray`

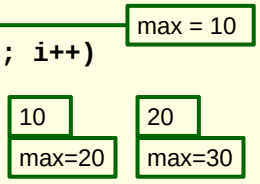
```
1) public static void printArray(int[] array)
2) {
3)   System.out.print("{ ");
4)   for (int i=0; i<array.length; i++)
5)   {
6)     System.out.print(array[i] +" ");
7)   }
8)   System.out.println("}");
9) }
```

Why make this **`public`**?

9

## Quiz: find max

```
1) int[] numList = {10, 5, 20, 13, 30, 15};
2)
3) int max = numList[0];
4) for (int i=1; i<numList.length; i++)
5) { if (numList[i] > max)
6)   { System.out.print(max+" ");
7)     max=numList[i];
8)   }
9) }
10)System.out.println();
```




What is the output of the above code segment?

a) **30**
b) **10 20**
c) **10 20 30**
d) **10 5 20 13 30**
e) **10 5 20 13 30 15**

10

## Find The Syntax Error

```
1) int[] numList = {10, 5, 20, 13, 30, 15};
2)
3) int max = numList;
4)
5)
6) for (int i=1; i<numList.length; i++)
7) {
8)   if (numList[i] > max)
9)   {
10)    System.out.print(max+" ");
11)    max=numList[i];
12)  }
13)}
14)System.out.println();
```

Type mismatch: Cannot Convert form **int[]** to **int**

11

## Quiz: find max

```
1) int[] numList = {33, 27, 55, 72, 18};
2)
3) int max = numList[0];
4) for (int i=1; i<numList.length; i++)
5) { System.out.print(i + "(" +max+") ");
6)   if (numList[i] > max)
7)   { max=numList[i];
8)   }
9) }
10)System.out.println();
```

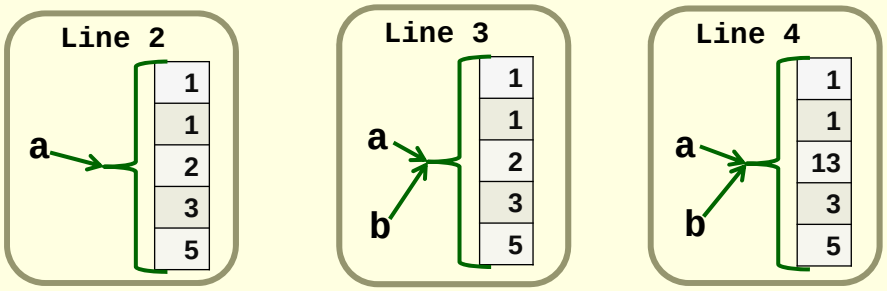
What is the output of the above code segment?

a) 0(33) 1(27) 2(55) 3(72) 4(18)
b) 1(33) 2(27) 3(55) 4(72) 5(18)
c) 1(33) 2(27) 3(55) 4(18)
d) 1(33) 2(33) 3(55) 4(72)
e) 1(33) 2(33) 3(55) 4(55)

12

## Array Reference Copy

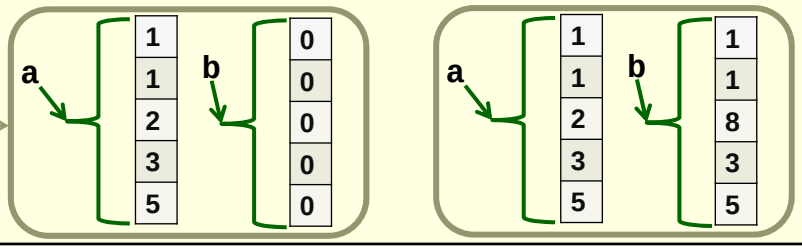
```
1)  public static void main(String[] args)
2)  { int[] a = {1, 1, 2, 3, 5};
3)    int[] b = a;
4)    b[2] = 13;
5)    System.out.println(
6)      a[1] + ", " + a[2] + ", " + b[2]);
7)  }
```




13

## Array Element Copy

```
1)  public static void main(String[] args)
2)  { int[] a = {1, 1, 2, 3, 5};
3)    int[] b = new int[5];
4)    for (i=0; i<a.length; i++)
5)    { b[i]=a[i];
6)    }
7)    b[2] = 8;
8)    System.out.println(
9)      a[1] + ", " + a[2] + ", " + b[2]);
10) }
```




14

## Quiz: Array Copying

```
1.  public static void main(String[] args)
2.  { int[] a = {1, 1, 2, 3, 5, 8};
3.    int[] b = a;
4.    b[2] = 7;
5.    System.out.println(
6.      a[1] + ", " + a[2] + ", " + b[2]);
7.  }
```

The output is:

a) 7, 7, 7
b) 1, 7, 7
c) 1, 1, 7
d) 1, 2, 7
e) 1, 1, 2

15

## Allocating Space for Arrays of Objects

```
1)  //Allocates space for three 32-bit integers.
2)  int[] x = new int[3];
3)
4)  //Allocates space for three references.
5)  JFrame[] frames = new JFrame[3];
6)
7)  for (int i = 0; i<3; i++)
8)  {
9)    x[i] = i;
10)
11)   //Create each of the JFrame instances.
12)   p[i] = new JFrame();
13)   frames[i].setBounds(i*10,i*10, 200, 200);
14)   frames[i].setVisible(true);
15) }
```

JFrame constructor NOT called.

JFrame constructor IS called.

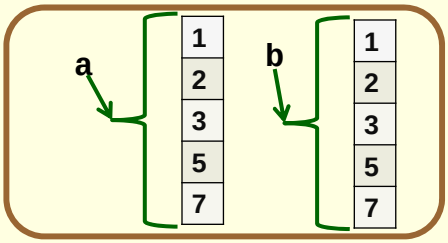
Without line 12, line 13 will cause a Null Pointer Exception.

16

## Comparing Array References

```
1. public static void main(String[] args)
2. { int[] a = {1, 2, 3, 5, 7};
3.   int[] b = {1, 2, 3, 5, 7};
4.   if (a==b)
5.   { System.out.println("yes");
6.   }
7.   else System.out.println("no");
8. }
```

Output: **no**




17

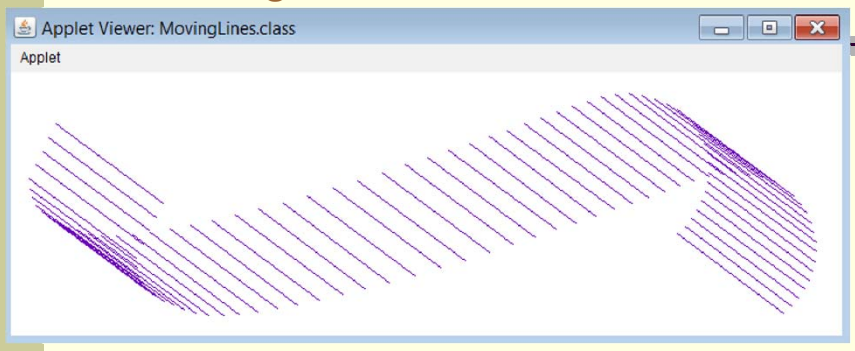
## Quiz: On Which line is the Syntax Error?

```
   public class SyntaxQuiz
   {
     public static double abs(double a)
     { if (a < 0.0) return -a;
       return a;
     }

     public static void main(String[] args)
     {
       double[] x = {1.2, -3.4, 7.1};
       int i = 1;
a)     System.out.println(abs(x[0]));
b)     System.out.println(abs(x[i]));
c)     System.out.println(abs(x));
d)     System.out.println(abs(i));
e)     System.out.println(abs(x[0] - x[2]));
     }
   }
```

18

## BouncingLines




1) Start with **`BouncingLine.java`**.
2) **`BouncingLine.java`** uses **`int x1, y1, x2, y2`** to maintain the location of the line's two endpoints.
3) Transform **`BouncingLine.java`** to use arrays **`int[] x1, y1, x2, y2`** to maintain a history of the line's endpoints.
4) After drawing **`NUM_LINES`**, each time step, erase the oldest line in the history and replace it with the line's new location.

19

## BouncingLines: imports

```
1) import java.awt.Color;
2) import java.awt.Graphics;
3) import java.awt.event.ActionEvent;
4) import java.awt.event.ActionListener;
5) import java.util.Random;
6) import javax.swing.Timer;
```

20

## BouncingLines: Class and Instance Fields

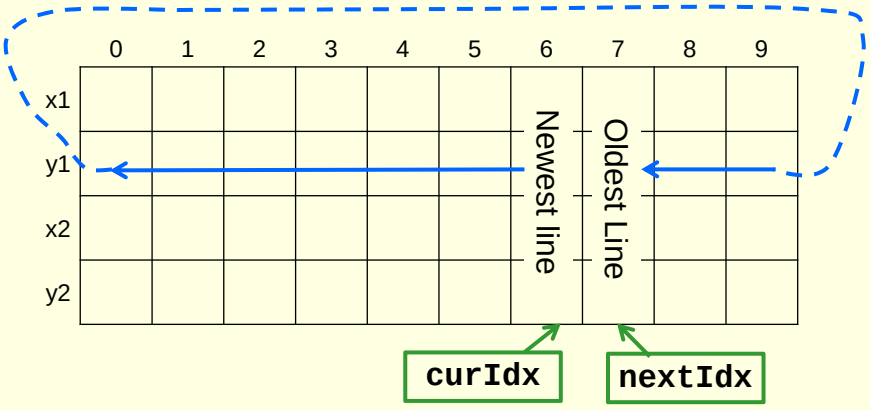
```
1)  public class BouncingLines implements ActionListener
2)  {
3)    private Random rand = new Random();
4)    private Timer myTimer;
5)    private Picture myPic;
6)    private Graphics canvas;
7)
8)    private static final int DRAW_WIDTH = 600;
9)    private static final int DRAW_HEIGHT = 600;
10)   private static final int NUM_LINES = 10;
11)
12)   private int[] x1 = new int[NUM_LINES];
13)   private int[] x2 = new int[NUM_LINES];
14)   private int[] y1 = new int[NUM_LINES];
15)   private int[] y2 = new int[NUM_LINES];
16)   private int curIdx = 0;
17)
18)   private int speedX1, speedX2, speedY1, speedY2;
```

Parallel Arrays

21

## Data Structure: Circular, Parallel Arrays

```
private int[] x1 = new int[NUM_LINES];
private int[] x2 = new int[NUM_LINES];
private int[] y1 = new int[NUM_LINES];
private int[] y2 = new int[NUM_LINES];
```




22

## BouncingLines: main

```
public static void main(String[] args)
{
  new BouncingLines();
}
```

23

## BouncingLines: Constructor

```
20) public BouncingLines()
21) {
22)   myPic = new Picture(DRAW_WIDTH, DRAW_HEIGHT);
23)   canvas = myPic.getOffScreenGraphics();
24)   canvas.setColor(Color.WHITE);
25)   canvas.fillRect(0, 0, DRAW_WIDTH, DRAW_HEIGHT);
26)   myPic.setTitle("Bounding Lines");
27)
28)   x1[0] = rand.nextInt(DRAW_WIDTH);
29)   y1[0] = rand.nextInt(DRAW_HEIGHT);
30)   x2[0] = rand.nextInt(DRAW_WIDTH);
31)   y2[0] = rand.nextInt(DRAW_HEIGHT);
32)
33)   speedX1 = rand.nextInt(25)-12;
34)   speedY1 = rand.nextInt(25)-12;
35)   speedX2 = rand.nextInt(25)-12;
36)   speedY2 = rand.nextInt(25)-12;
37)
38)   myTimer = new Timer(10, this); // miliseconds
39)   myTimer.start();
40) }
```

24

## BouncingLines: `actionPerformed` (1 of 5)

```
50) public void actionPerformed(ActionEvent arg0)
51) {
52)    //Let  nextIdx  be the index of the oldest line.
53)    int nextIdx = (curIdx+1) % NUM_LINES;
       :
55)    //Erase oldest line.
       :
60)    //Move newest line and store its endpoints at index where
       :      //    the oldest line was stored.
       :
70)    //If next line is out of bounds in the horizontal direction, then
       :      //    give it a random horizontal speed in the opposite direction.
       :      //If next line is out of bounds in the vertical direction, then
       :      //    give it a random vertical speed in the opposite direction.
       :
90)    //Update the current line index.
       :
93)    //Draw the new line
99) }
```

25

## BouncingLines: `actionPerformed` (2 of 5)

```
50) public void actionPerformed(ActionEvent arg0)
51) {
52)    //Let  nextIdx  be the index of the oldest line.
53)    int nextIdx = (curIdx+1) % NUM_LINES;
54)
55)    //Erase oldest line.
56)    canvas.setColor(Color.WHITE);
57)    canvas.drawLine(x1[nextIdx], y1[nextIdx],
58)                    x2[nextIdx], y2[nextIdx]);
```

26

## BouncingLines: **actionPerformed** (3 of 5)

```
60)  //Move newest line and store its endpoints at index
61)  //    where the oldest line was stored.
62)  x1[nextIdx] = x1[curIdx] + speedX1;
63)  y1[nextIdx] = y1[curIdx] + speedY1;
64)
65)  x2[nextIdx] = x2[curIdx] + speedX2;
66)  y2[nextIdx] = y2[curIdx] + speedY2;
67)
68)
```

27

## BouncingLines: **actionPerformed** (4 of 5)

```
70)  //If next line is out of bounds in the horizontal direction, then
71)  //    give it a random horizontal speed in the opposite direction.
72)  //If next line is out of bounds in the vertical direction, then
73)  //    give it a random vertical speed in the opposite direction.
74)  if (x1[nextIdx] < 0) speedX1 = rand.nextInt(12)+1;
75)  if (y1[nextIdx] < 0) speedY1 = rand.nextInt(12)+1;
76)
77)  if (x1[nextIdx]>DRAW_WIDTH) speedX1 =-(rand.nextInt(12)+1);
78)  if (y1[nextIdx]>DRAW_HEIGHT) speedY1=-(rand.nextInt(12)+1);
79)
80)
81)
82)  if (x2[nextIdx] < 0) speedX2 = rand.nextInt(12)+1;
83)  if (y2[nextIdx] < 0) speedY2 = rand.nextInt(12)+1;
84)
85)  if (x2[nextIdx]>DRAW_WIDTH) speedX2 =-(rand.nextInt(12)+1);
86)  if (y2[nextIdx]>DRAW_HEIGHT) speedY2=-(rand.nextInt(12)+1);
87)
```

28

## BouncingLines: **actionPerformed** (5 of 5)

```
90)   //Update the current line index.
91)   curIdx = nextIdx;
92)
93)   //Draw the new line
94)   canvas.setColor(Color.RED);
95)   canvas.drawLine(x1[curIdx], y1[curIdx],
96)                   x2[curIdx], y2[curIdx]);
97)   myPic.repaint();
98)
99)}
```

29