Arrays Part 1
Topic 19

"Should array indices start at 0 or 1? My compromise of 0.5 was rejected without, I thought, proper consideration. "
- *Stan Kelly-Bootle*

"Arrays are our friends. They're here to help."
-unknown

Based on slides for Building Java Programs by Reges/Stepp, found at http://faculty.washington.edu/stepp/book/

A problem

- In the file processing case study we found all lines in an input file that contained a search phrase
- The file was processed line by line
- When a line with the search phrase was found the result were output right then, before moving on.
- We couldn't save the input lines for later use
  - Why not?

A problem we can't solve (yet)

- 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 = 44.57142857142857
4 days were above average.
```

- We need access to the temperatures once to compute the average, and then again to tell how many were above average.

Why the problem is tough

- We appear to need each input value twice:
  - once to compute the average
  - a second time to count how many were above average
- We could examine each value twice if we could read them into variables.
- However, we don't know in advance how many variables we'll need, because we don't know how many days' weather are coming until the program is running.
- We need a way to declare many variables' worth of memory in a single step.

Another tough problem

- Given a file of integer exam scores, such as:

```
82
66
79
63
83
81
```

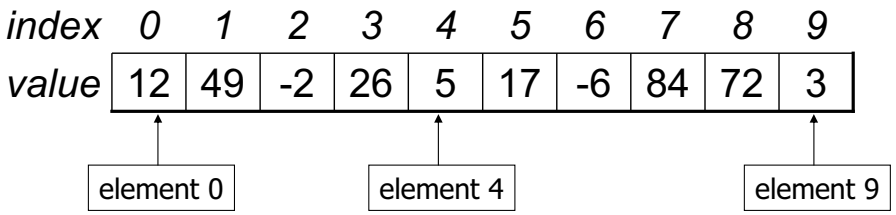
- Write a program that will print a text histogram of stars indicating the number of students who earned each unique exam score.

```
85: *****
86: ***********
87: ***
88: *
91: ****
```


Arrays

- **array**: A single variable that can store many values of the same type.
  - **element**: One value in an array.
  - **index**: A 0-based integer, which is used to access an element from an array.

- We usually draw an array as a row or column of boxes.
  - Example: an array of ten integers



Array declaration

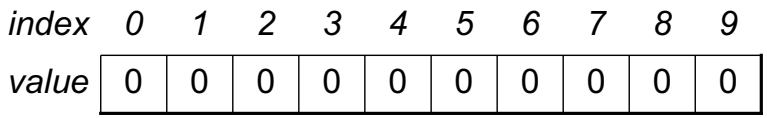
- Declaring/initializing an array:

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

  - Example:

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

  - The length of the array is specified between [ ] brackets.

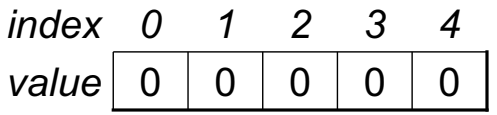


- The array's length can be any expression, such as a constant or variable's value.
  - Example:

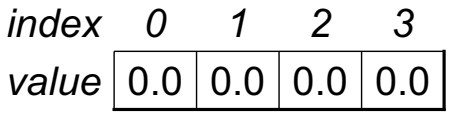
```
int x = 2 * 3 + 1;
int[] numbers = new int[x % 5 + 2];
```


Array initial contents

- When arrays are initially constructed, every element has a 0-equivalent value.
  - `int`: `0`
  - `double`: `0.0`
  - `boolean`: `false`
  - `char`: `'\0'` (the "null character")
  - `String` or other object: `null` (`null` means "no object")



An array of integers



An array of real numbers

Accessing array elements

- Assigning a value to an array element:
  <array name> `[` ***<index>*** `] = ` ***<value>*** `;`
  - Example:

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

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

- Using an array element's value in an expression:
  <array name> `[` ***<index>*** `]`
  - Example:

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


Out-of-bounds indices

- The only indices that are legal to access in an array are those in the range of **0** to the **array's length - 1**
  - Reading or writing any index outside this range will crash your program with an `ArrayIndexOutOfBoundsException`.
- Example:

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

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

Arrays of other types

- Arrays can contain other types, such as double.
  - Example:

```
double[] results = new double[6];
results[2] = 3.4;
results[5] = -0.5;
```

index	*0*	*1*	*2*	*3*	*4*	*5*
value	0.0	0.0	**3.4**	0.0	0.0	**-0.5**

  - Example:

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

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

Accessing array elements

- A longer example of accessing and changing elements:

```
int[] numbers = new int[8];
numbers[1] = 4;
numbers[4] = 99;
numbers[7] = 2;

int x = numbers[1];
numbers[x] = 44;
numbers[ numbers[7] ] = 11;
```

x	4

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
numbers	0	4	11	0	44	0	0	2

Arrays and for loops

- Arrays are very commonly used with `for` loops that pass over each element and process it in some way:
  - Example (print each element of an array):

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

  - Output (when used on array from previous slide):

```
0 4 11 0 44 0 0 2
```


More arrays and for loops

- Sometimes we assign each array element a value in a for loop.
  - Example:

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

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

- What values would be stored into the array after this code?

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

value								

- Notice that the code in this slide refers to the array's length of 8 many times. What's bad about this?

The .length field

- An array has a field named `.length` that returns its total number of elements.
  - General syntax:
    <array name> `.length`
    - Notice that it doesn't have parentheses like a String's `.length()`.
- Example:

```
for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + " ");
}
```

  - Output:

```
0 1 4 9 16 25 36 49
```

- What expression refers to the last element of the array? The middle element?

A multi-counter problem

- Imagine that we want to examine a large integer and count the number of occurrences of every digit from 0 through 9.
  - Example: the number 229231007 contains two 0s, one 1, three 2s, one 7, and one 9.
  - How can we do it?
- We need to examine each digit of the large integer and count how many times we've seen that digit.
  - This will require counters for each of the values 0--9.
- We could declare 10 counter variables for this, or (preferred) we could use an array of size 10.
  - The element with index *i* will store the counter for digit value *i*.

Creating an array of tallies

- The following code builds an array of digit counters:

```
int num = 229231007;
int[] counts = new int[10];
while (num > 0) {
    int digit = num % 10;
    counts[digit]++;
    num = num / 10;
}
```

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

  - You can watch the array build itself by running the code in the *Jeliot* Java program visualizer: http://www.cs.joensuu.fi/jeliot/index.php

Histogram problem

- Given a file of integer exam scores, such as:

```
82
66
79
63
83
81
```

- Write a program that will print a histogram of stars indicating the number of students who earned each unique exam score.

```
85: *****
86: ************
87: ***
88: *
91: ****
```

- Time permitting, try graphing the data with DrawingPanel.

Why are arrays useful?

- We can use arrays to store a large amount of data without declaring many variables.
  - Example: Read in a file of 1000 numbers, then print out the numbers in reverse order.
- Arrays can help us to group related data into elements.
  - Example: For a given school exam, open a file full of exam scores and count up how many students got each score from 0 through 100.
- Arrays let us hold on to data and access it in random order.
  - Example: Read a file full of babies' names, store each name's data as an element in a large array, and then examine it later to find many names that the user types.

Array initialization shortcut

- Quick array initialization, general syntax:

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

  - Example:

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

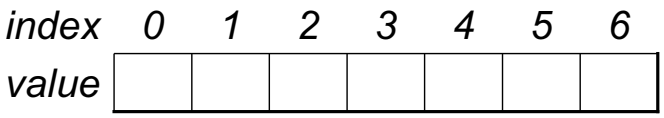
index	0	1	2	3	4	5	6
value	12	49	-2	26	5	17	-6

  - This syntax is useful when you know in advance what the array's elements will be.
  - You don't explicitly specify the array's size when you use this syntax -- the Java compiler figures this out by looking at the number of values written.

Array practice problem

- What are the contents of the array after the following code?

```
int[] a = {2, 5, 1, 6, 14, 7, 9};
for (int i = 1; i < a.length; i++) {
    a[i] += a[i - 1];
}
```



Review: primitive variables

- We now need to examine some important differences between the behavior of objects/arrays and primitive values.
- We saw with primitive variables that modifying the value of one variable does not modify the value of another.
- When one variable is assigned to another, the value is copied.
  - Example:

```
int x = 5;
int y = x;    // x = 5, y = 5
y = 17;       // x = 5, y = 17
x = 8;        // x = 8, y = 17
```


Reference variables

- However, objects behave differently than primitives.
  - When working with objects, we have to understand the distinction between an object, and the variable that stores it.
  - Variables of object types are called **reference variables**.
  - Reference variables do not actually store an object; they store the address of an object's location in the computer memory.
  - If two reference variables are assigned to refer to the same object, the object is *not* copied; both variables literally share the same object. Calling a method on either variable will modify the same object.
  - Example:

```
DrawingPanel p1 = new DrawingPanel(80, 50);
DrawingPanel p2 = p1;   // same window
p2.setBackground(Color.CYAN);
```


References example

- When `panel2` refers to the same object as `panel1`, modifying either variable's background color will affect the same window:

```
DrawingPanel panel1 = new DrawingPanel(80, 50);
DrawingPanel panel2 = new DrawingPanel(80, 50);
DrawingPanel panel3 = new DrawingPanel(80, 50);
panel1.setBackground(Color.RED);
panel2.setBackground(Color.GREEN);
panel3.setBackground(Color.BLUE);

panel2 = panel1;
panel2.setBackground(Color.MAGENTA);
```

Modifying parameters

- When we call a method and pass primitive variables' values as parameters, it is legal to assign new values to the parameters inside the method.
  - But this does not affect the value of the variable that was passed, because its value was copied.
  - Example:

```
public static void main(String[] args) {
    int x = 1;
    foo(x);
    System.out.println(x);   // output: 1
}

public static void foo(int x) {
    x = 2;
}
```

value 1 is copied into parameter

parameter's value is changed to 2
(variable `x` in `main` is unaffected)

Objects as parameters

- When an object is passed as a parameter, it is not copied. It is shared between the original variable and the method's parameter.
  - If a method is called on the parameter, it *will* affect the original object that was passed to the method.
  - Example:

```
public static void main(String[] args) {
    DrawingPanel p = new DrawingPanel(80, 50);
    p.setBackground(Color.YELLOW);
    bg(p);
}

public static void bg(DrawingPanel panel) {
    panel.setBackground(Color.CYAN);
}
```

  - Note: This is the reason that it works when you pass the `Graphics g` as a parameter to a method, because it is drawing with the same pen object onto the same window.

Arrays as parameters

- An array can also be passed as a parameter.
  - Example:

```
int[] iq = new int[3];
iq[0] = 126;
iq[1] = 167;
iq[2] = 95;
int max = getMaxValue(iq);
System.out.println("Max = " + max);
...

public static int getMaxValue(int[] array) {
    int max = array[0];
    for (int i = 1; i < array.length; i++) {
        if (array[i] > max) {
            max = array[i];
        }
    }
    return max;
}
```

  - Output:

```
Max = 167
```


Arrays as parameters, contd.

- When an array is passed as a parameter, it is passed as a *reference* (similar to objects).
  - If the method modifies elements of its array parameter, the changes will also be seen in the original array.
  - Example:

```
public static void main(String[] args) {
    int[] iq = new int[3];
    iq[0] = 126;
    iq[1] = 167;
    iq[2] = 95;
    destroy(iq);
    System.out.println(iq[0] + " " + iq[1]);
}

public static void destroy(int[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] = 0;  // ha ha!
    }
}
```

  - Output:

```
0 0
```

Array practice problems

1. Write a method to find the maximum value in an array of ints
2. Write a method to find the minimum value in an array of ints
3. Write a method to find the minimum and maximum value in an array of ints
4. Write a method to find how many times a given values appears in an array of ints
5. Write a method to determine the first index of a given value in an array of ints
6. Write a method to determine the last index of a given value in an array of ints
7. Write a method to determine how many times a given character occurs in an array of Strings

More array problems

1. Write a method to replace all negative values in an array with the maximum value in the array
2. Write a method that takes an array of ints and returns an array of ints. The returned array contains the same elements as the original array in the same order, except all elements less than a given value are removed
3. Write a method to reverse the elements of an array
4. Write a method to determine if two arrays overlap at a given index
5. Write a method to find the maximum overlap of two arrays
6. Write a method to compute the mean of an array of ints
7. Write a method to compute the standard deviation of an array of ints

More array problems

1. Write a method named `equal` that accepts 2 arrays of integers as its parameters and returns whether those two arrays contain exactly the same elements in the same order.
2. Write a method named `print` that accepts an array of integers as its parameter and prints the elements of the array in the following format:
   `{7, -2, 46, 3, 55}`
3. Write a method named `roundAll` that accepts an array of `double`s as its parameter and modifies each element of the array so that it is rounded to the nearest whole number.