Building Java Programs

Lab 7: Ch. 7: Arrays

lab document created by Marty Stepp and Stuart Reges

Today's lab

Goals for today:

- use arrays for storing lists of data
- practice using `for` loops to traverse and process array data
- pass arrays as parameters and returns from methods
- explore the idea of reference parameters
- Where you see this icon, you can click it to check the problem in Practice-It! 

Exercise 1: Array declaration syntax

Which of the following choices is the correct syntax for declaring/initializing an array of integers?

a. ○ `int a[10];`

b. ○ `[]int a = [10]int;`

c. ○ `int[10] a = new int[10];`

d. ○ `int[] a = new int[10];`

e. ○ `int a[10] = new int[10];`

Exercise 2: Quick initialization syntax

Which of the following choices is the correct syntax for quickly declaring/initializing an array of integers to store a particular list of values?

a. ○ `int[] a = {17, -3, 42, 5, 9, 28};`

b. ○ `int a {17, -3, 42, 5, 9, 28};`

c. ○ `int[] a = new int[6] {17, -3, 42, 5, 9, 28};`

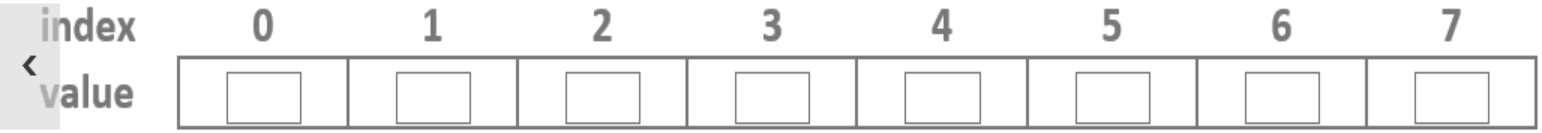
d. ○ `int[6] a = {17, -3, 42, 5, 9, 28};`

e. ○ `int[] a = new {17, -3, 42, 5, 9, 28} [6];`

Exercise 3: Array code tracing

Fill in the array with the values that would be stored after the code executes:

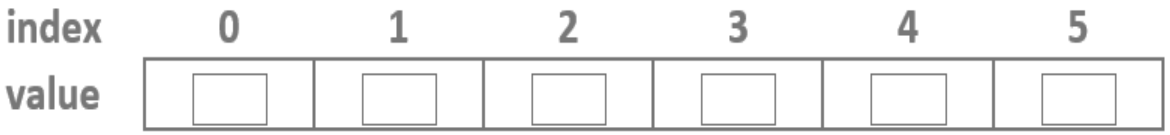
```
int[] data = new int[8];
data[0] = 3;
data[7] = -18;
data[4] = 5;
data[1] = data[0];

int x = data[4];
data[4] = 6;
data[x] = data[0] * data[1];
```



Exercise 4: Array code tracing 2

Fill in the array with the values that would be stored after the code executes:

```
int[] list = {2, 18, 6, -4, 5, 1};
for (int i = 0; i < list.length; i++) {
    list[i] = list[i] + (list[i] / list[0]);
}
```



Exercise 5: PromptNumbers

- **Download** the following file PromptNumbers.java to your machine and open it with jGrasp.
- The program is supposed to prompt the user to enter several integers, store them into an array, then print those integers back out in forwards and backwards order. Finish the program so that it runs properly. Use an array to make your program flexible enough that it will work no matter how many integers the user wants to type.

Arrays as parameter/return (declare)

```
public static type name(type[] name) {   // pass array parameter
```

```
public static type[] name(parameters) {   // return array
```

- Arrays can be passed as parameters and returned from methods.
- This method takes an array of doubles, and returns a new array of rounded ints:

```
public static int[] roundAll(double[] realNums) {
    int[] roundedNums = new int[realNums.length];
    for (int i = 0; i < realNums.length; i++) {
        roundedNums[i] = (int) Math.round(realNums[i]);
    }
    return roundedNums;
}
```

Arrays as parameter/return (call)

- Below is an example usage of the `roundAll` method from the previous slide:

```
import java.util.*; // to use Arrays

public class MyProgram {
    public static void main(String[] args) {
        double[] realNumbers = {5.5, 7.31, 8.09, -3.234234, 2.0, 0.0};
        int[] roundedNumbers = roundAll(realNumbers);
        System.out.println(Arrays.toString(roundedNumbers));
    }
    ...
}

// Output: [5, 7, 8, -3, 2, 0]
```

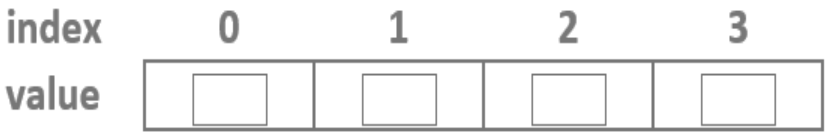
Arrays class methods

Method name	Description
`Arrays.binarySearch(`***array***`, `***value***`)`	returns index of value in a sorted array (< 0 if not found)
`Arrays.copyOf(`***array***`, `***length***`)`	returns a new copy of an array
`Arrays.equals(`***array1***`, `***array2***`)`	returns true if the two arrays contain same elements
`Arrays.fill(`***array***`, `***value***`)`	sets every element to the given value
`Arrays.sort(`***array***`)`	arranges the elements into sorted order
`Arrays.toString(`***array***`)`	returns a string for the array, such as "[10, 30, -25, 17]"

Exercise 6: Array reference mystery

What values are stored in the array at the comment in `main`? Note that the `incrementAll` returns `void`, but does take an `int[]` parameter.

```
public class ArrayReference {
    public static void main(String[] args) {
        int[] nums = {2, 4, -1, 3};
        incrementAll(nums);

        // HERE!
    }

    public static void incrementAll(int[] data) {
        for (int i = 0; i < data.length; i++) {
            data[i]++;
        }
    }
}
```



Exercise 7: array mystery

Suppose that each array at right were passed as a parameter to the `mystery` method below. Fill in the boxes with the array contents after each method call.

```
public static void mystery(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        if (a[i] < a[i + 1]) {
            a[i] = a[i + 1];
        }
    }
}
```

```
int[] a1 = {2, 4};
mystery(a1);
```

```
int[] a2 = {1, 3, 6};
mystery(a2);
```

```
int[] a3 = {7, 2, 8, 4};
mystery(a3);
```

```
int[] a4 = {5, 2, 7, 2, 4};
mystery(a4);
```

```
int[] a5 = {2, 4, 6, 3, 7, 9};
mystery(a5);
```

Exercise 8: jGRASP Debugger

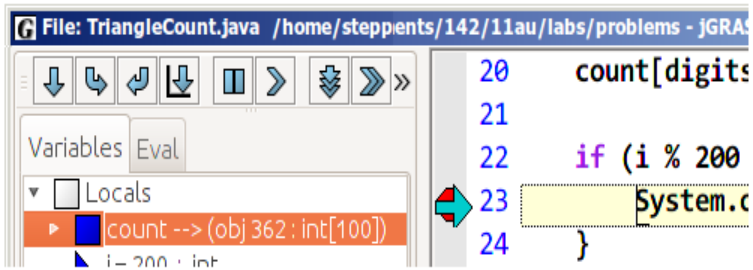
- **Download** the file TriangleCount.java to your machine and open it with jGrasp.
- This program explores patterns of digits for what are called the *triangle numbers*. The program counts how many times a triangle number ends in a particular 2-digit sequence. For example, the 15th triangle number is 120, which ends in the 2-digit sequence 20.
- **Run** the program and you will see a pattern: Most counts are 0 and most of the rest are exactly 200. **Which 2-digit sequences have a count of 500?**

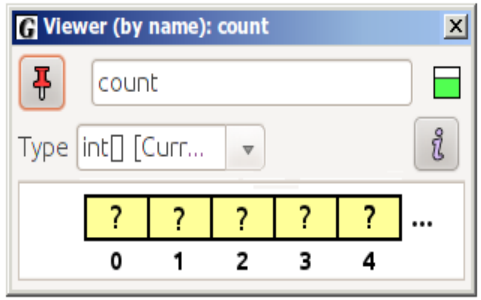
continued on the next slide...

Exercise 8 - jGRASP Debugger

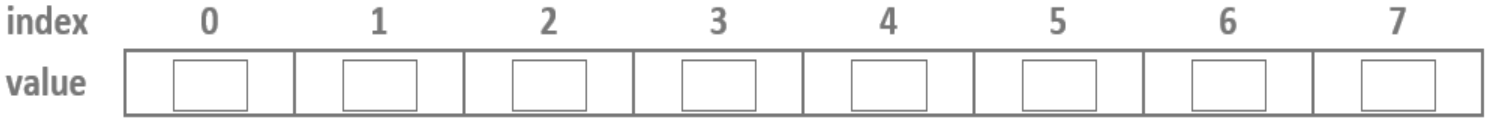
- The program prints a progress dot every 200 iterations. **Set a breakpoint** on the `System.out.print` inside the `if` on line 23 and **debug** the program.

- Once the program pauses, **click and drag** the array at left named `count`, and drop it on the main code window. This will show a view of its contents.





- What are the **first eight element values** stored in the `count` array?



continued on the next slide...

Exercise 8 - jGRASP Debugger

- Leave the array viewer open and click the **Resume** button  . *(This runs until the second time the `System.out.print` is about to execute, then stops.)*

- What values do the **first 8 array elements** have now?

index	0	1	2	3	4	5	6	7
value								

- This is a very long array with 100 total values.
  Scroll to the far right in the array viewer. What values do the **last 8 elements** have?

index	92	93	94	95	96	97	98	99
value								

Exercise 9: max errors

```
1  // Returns the largest value in the given array.
2  public static int max(int data[10]) {
3      int max = 0;
4      for (int i = 0; i < data[].length(); i++) {
5          if (array[i] > max) {
6              max = array[i];
7          }
8      }
9      return max[];
10 }
```

The above attempted solution to Practice-It problem "max" has 7 problems. Open Practice-It from the link above, copy/paste this code into it, and fix the errors. Complete the code so that it passes the test cases.

Exercise 9 - answer

1. line 2: should not write `[10]` after parameter name; should write `[]` (without length) after `int`
2. line 3: starting `max` at `0` won't work if the array is all negative. Should initialize `max` variable to be `data[0]` and start `for` loop at index `1`
3. line 4: should not write `[]` after `data` here
4. line 4: should not write `()` after `length` here
5. line 5: `array` should be `data`
6. line 6: `array` should be `data`
7. line 9: should not write `[]` after `max` here

Exercise 9 - solution

```
1  // Returns the largest value in the given array.
2  public static int max(int[] data) {
3      int max = data[0];
4      for (int i = 1; i < data.length; i++) {
5          if (data[i] > max) {
6              max = data[i];
7          }
8      }
9      return max;
10 }
```

Exercise 10: PromptNumbers2

- **Modify** your `PromptNumbers.java` program from a previous exercise.
- **Add a method** that accepts an array as a parameter and prints the elements of that array in forward order.
- Then **add a second method** that accepts an array as a parameter and prints the elements of that array in *backward* order.

Exercise 10 - answer

```
public class PromptNumbers2 {
    public static void main(String[] args) {
        int count = console.nextInt();
        int[] nums = new int[count];
        ...
        System.out.println("Your numbers in forward order:");
        printForward(nums);

        System.out.println("Your numbers in backward order:");
        printBackward(nums);
    }

    // Prints the elements of the given array in forward order.
    public static void printForward(int[] a) {
        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i]);
        }
    }
    // Prints the elements of the given array in backward order.
    public static void printBackward(int[] a) {
        for (int i = a.length - 1; i >= 0; i--) {
            System.out.println(a[i]);
        }
    }
}
```

Exercise 11: minGap

Write a method named `minGap` that accepts an integer array as a parameter and returns the minimum 'gap' between adjacent values in the array. The gap between two adjacent values in a array is defined as the second value minus the first value. For example, suppose a variable called `array` is an array of integers that stores the following sequence of values:

```
int[] array = {1, 3, 6, 7, 12};
```

The first gap is 2 (3 - 1), the second gap is 3 (6 - 3), the third gap is 1 (7 - 6) and the fourth gap is 5 (12 - 7). Thus, the call of `minGap(array)` should return `1` because that is the smallest gap in the array. If you are passed an array with fewer than 2 elements, you should return `0`.

Click on the check-mark above to try out your solution in Practice-it!

Exercise 12: `percentEven`

Write a method named `percentEven` that accepts an array of integers as a parameter and returns the percentage of even numbers in the array as a real number. For example, if a variable named `nums` refers to an array of the elements `{6, 2, 9, 11, 3}`, then the call of `percentEven(nums)` should return `40.0`. If the array contains no even elements or no elements at all, return `0.0`.

Click on the check-mark above to try out your solution in Practice-it!

Exercise 13: swapAll

Write a method named `swapAll` that accepts two arrays of integers as parameters and swaps their entire contents. You may assume that the arrays passed are not null and are the same length.

For example, if the following arrays are passed:

```
int[] a1 = {11, 42, -5, 27, 0, 89};
int[] a2 = {10, 20, 30, 40, 50, 60};
swapAll(a1, a2);
```

After the call, the arrays should store the following elements:

```
a1: {10, 20, 30, 40, 50, 60}
a2: {11, 42, -5, 27, 0, 89}
```

Exercise 14: stretch

Write a method named `stretch` that accepts an array of integers as a parameter and returns a new array twice as large as the original, replacing every integer from the original array with a pair of integers, each half the original. If a number in the original array is odd, then the first number in the new pair should be one higher than the second so that the sum equals the original number.

For example, if a variable named `list` refers to an array storing the values `{18, 7, 4, 24, 11}`, the call of `stretch(list)` should return a new array containing `{9, 9, 4, 3, 2, 2, 12, 12, 6, 5}`. (The number 18 is stretched into the pair 9, 9, the number 7 is stretched into 4, 3, the number 4 is stretched into 2, 2, the number 24 is stretched into 12, 12 and the number 11 is stretched into 6, 5.)

Click on the check-mark above to try out your solution in Practice-it!

Exercise 15: copyRange

Write a method named `copyRange` that takes as parameters two arrays `a1` and `a2`, two starting indexes `i1` and `i2`, and a length `l`, and copies the first `l` elements of `a1` starting at index `i1` into array `a2` starting at index `i2`.

For example, if the following arrays are declared:

```
int[] a1 = {10, 20, 30, 40, 50, 60};
int[] a2 = {91, 92, 93, 94, 95, 96};
copyRange(a1, a2, 0, 2, 3);
```

After the preceding call, the contents of `a2` would be `{91, 92, 10, 20, 30, 96}`. You may assume that the parameters' values are valid, that the arrays are large enough to hold the data, and so on.

Exercise 16: equals

Write a method named `equals` that accepts two arrays of integers as parameters and returns `true` if they contain exactly the same elements in the same order, and `false` otherwise. Note that the arrays might not be the same length; if the lengths differ, return `false`. Do not call `Arrays.equals` in your solution.

For example, if the following arrays are declared:

```
int[] a1 = {10, 20, 30, 40, 50, 60};
int[] a2 = {10, 20, 30, 40, 50, 60};
int[] a3 = {20, 3, 50, 10, 68};
```

The call `equals(a1, a2)` returns `true` but the call `equals(a1, a3)` returns `false`.

If you finish them all...

If you finish all the exercises, try out our **Practice-It** web tool. It lets you solve Java problems from our *Building Java Programs* textbook.

You can view an exercise, type a solution, and submit it to see if you have solved it correctly.

Choose some problems from the book and try to solve them!