CS101

Java Programming

Winter 2013

Thursdays 10:00-12:00noon

RF Academy

Announcements

- Read Section 7.3
- Any Questions?!?!

Review of arrays

- In each of the following, **`e`** can be any expression that produces an **`int`**
  - Variable referring to an array: **`type[] name;`**
  - Array creation: **`new type[e]`**
  - Access array element: **`name[e]`**
  - Update array element: **`name[e] = …;`**
  - Get array's length: **`name.length`**

Arrays as parameters

- Declaration:

```
public static type methodName(type[] name) {
```

  - Example:

```
public static double average(int[] numbers) {
```

- Call:

```
methodName(arrayName);
```

  - Example:

```
int[] scores = {13, 17, 12, 15, 11};
double avg = average(scores);
```

Note: there is no use of square brackets [ ] at the call site

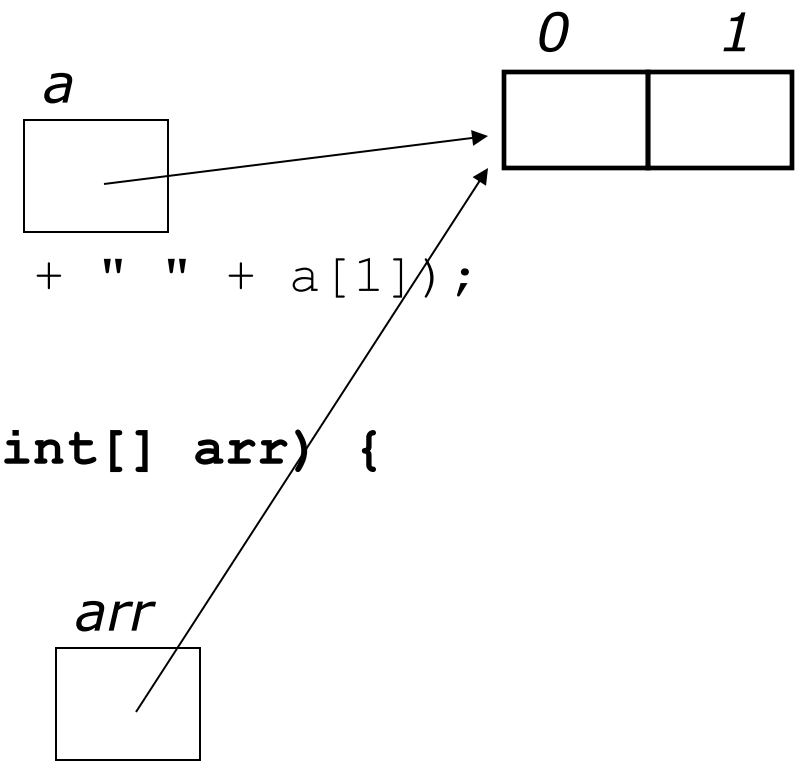
The truth about `new`

- `new` doesn't actually return an array (or an object)
- It returns a "URL" for a new array (or a new object)
  - We call them "references" (or "addresses" or "pointers")
- A variable never "holds" an array (or an object)
  - It always holds a reference to an array (or an object)
  - So when we copy the reference (think URL), there are now two references *to the same array*
- `a[i] = 42;` follows the reference and updates the array
  - So any other references to the same array see the change

That explains everything

```
public static void main(String[] args) {
    int[] a = new int[2];
    a[0] = 42;
    a[1] = 64;
    swap(a);
    System.out.println(a[0] + " " + a[1]);
  }

  public static void swap(int[] arr) {
      int temp = arr[0];
      arr[0] = arr[1];
      arr[1] = temp;
  }
```



– Arrays are passed as parameters by *reference.*
  • Changes made in the method are also seen by the caller.

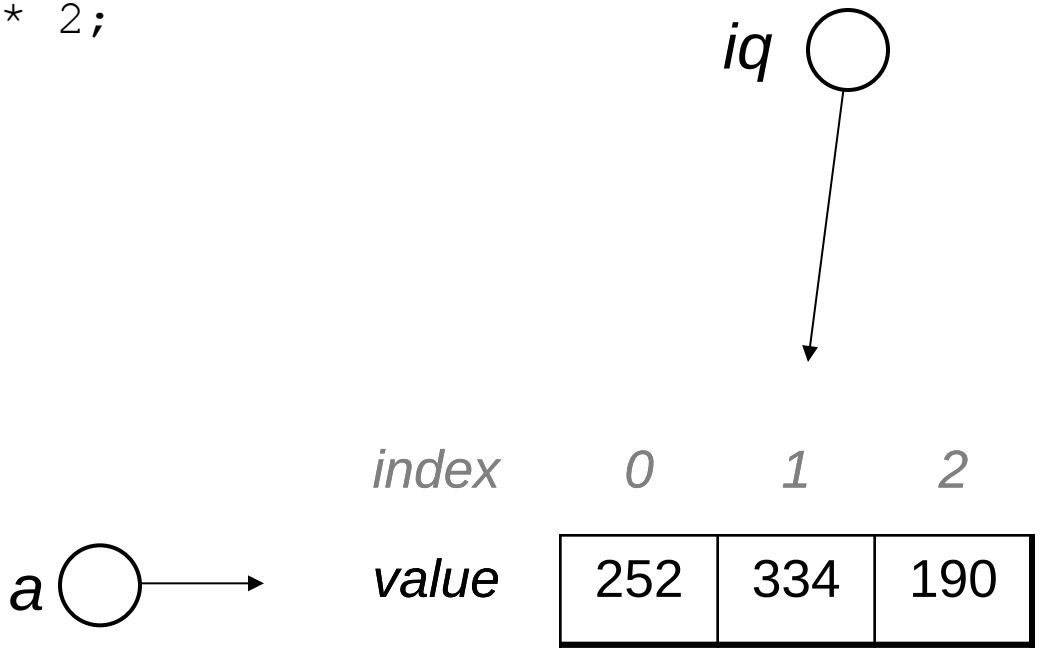
A useful parameter example

```
public static void main(String[] args) {
    int[] iq = {126, 167, 95};
    doubleAll(iq);
    System.out.println(Arrays.toString(iq));
}

public static void doubleAll(int[] a) {
    for (int i = 0; i < a.length; i++) {
        a[i] = a[i] * 2;
    }
}
```

- Output:

```
[252, 334, 190]
```



Array parameter questions

- Write a method `swap` that accepts an array of integers and two indexes and swaps the elements at those indexes.

```
int[] a1 = {12, 34, 56};
swap(a1, 1, 2);
System.out.println(Arrays.toString(a1));  // [12, 56, 34]
```

- Write a method `swapAll` that accepts two arrays of integers as parameters and swaps their entire contents.
  - Assume that the two arrays are the same length.

```
int[] a1 = {12, 34, 56};
int[] a2 = {20, 50, 80};
swapAll(a1, a2);
System.out.println(Arrays.toString(a1));  // [20, 50, 80]
System.out.println(Arrays.toString(a2));  // [12, 34, 56]
```

Array parameter answers

```
// Swaps the values at the given two indexes.
public static void swap(int[] a, int i, int j) {
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

// Swaps the entire contents of a1 with those of a2.
// Assumes the two arrays have the same length.
public static void swapAll(int[] a1, int[] a2) {
    for (int i = 0; i < a1.length; i++) {
        int temp = a1[i];
        a1[i] = a2[i];
        a2[i] = temp;
    }
}
```

Output parameters

- **output parameter**: An array (or any object) passed as a parameter that has its contents altered by the method.
  - We can pass in an array to a method and the method can change its contents in useful ways for us.
  - Examples: The methods `Arrays.fill` and `Arrays.sort`.

```
int[] nums = {4, -1, 2, 7, 3, 6};
System.out.println(Arrays.toString(nums));
Arrays.sort(nums);   // modifies contents of nums
System.out.println(Arrays.toString(nums));
Arrays.fill(nums, 42);
System.out.println(Arrays.toString(nums));
```

Output:

```
[4, -1, 2, 7, 3, 6]
[-1, 2, 3, 4, 6, 7]
[42, 42, 42, 42, 42, 42]
```

Output parameters

- **output parameter**: The discussion of output parameters does not apply to String objects. This is because, in Java, String objects are immutable. Instead, all our String operations create & return new strings. The original String object continues to exists, unchanged.

Array parameter questions

- Write a method named `average` that accepts an array of integers as its parameter and returns the average of the values in the array.
  - We already did this!
- Write a method named `contains` that accepts an array of integers and a target integer value as its parameters and returns true/false indicating whether the array contains the target value as one of its elements.
- Write a method named `roundAll` that accepts an array of `double`s as its parameter and modifies each array element by rounding it to the nearest whole number.

Array parameter answers

```
public static double average(int[] number) {
    int sum = 0;
    for (int i = 0; i < number.length; i++) {
        sum += number[i];
    }
    return (double) sum / number.length;
}

public static boolean contains(int[] value, int target) {
    for (int val : value) {  // uses a for-each loop instead
        if (val == target) {
            return true;
        }
    }
    return false;
}

public static void roundAll(double[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] = Math.round(array[i]);
    }
}
```

Arrays as return value (declaring)

```
public static type[] methodName(parameters) {
```

– Example:

```
public static int[] countDigits(int n) {
    int[] counter = new int[10];
    while (n > 0) {
        int digit = n % 10;
        counter[digit]++;
        n = n / 10;
    }
    return counter;
}
```

Arrays as return value (calling)

type`[]` **name** `=` **methodName**`(`**parameters**`);`

– Example:

```
public static void main(String[] args) {
    int[] tally = countDigits(229231007);
    System.out.println(Arrays.toString(tally));
}
```

Output:

```
[2, 1, 3, 1, 0, 0, 0, 1, 0, 1]
```

- Note: no need to initialize `tally` with a `'new'` operation
  - The `'new'` operation is done in `countDigits`

Array return (declare)

- Example #2:

```
// Returns a new array with two copies of each value.
// Example: [1, 4, 0, 7] -> [1, 1, 4, 4, 0, 0, 7, 7]
public static int[] stutter(int[] numbers) {
    int[] result = new int[2 * numbers.length];
    for (int i = 0; i < numbers.length; i++) {
        result[2 * i]     = numbers[i];
        result[2 * i + 1] = numbers[i];
    }
    return result;
}
```

Array return (call)

- Example #2:

```
public class MyProgram {
    public static void main(String[] args) {
        int[] iq = {126, 84, 149, 167, 95};
        int[] stuttered = stutter(iq);
        System.out.println(Arrays.toString(stuttered));
    }
    ...
```

- Output:

```
[126, 126, 84, 84, 149, 149, 167, 167, 95, 95]
```

Array return question

- Write a method `merge` that accepts two arrays of integers and returns a new array containing all elements of the first array followed by all elements of the second.

```
int[] a1 = {12, 34, 56};
int[] a2 = {7, 8, 9, 10};

int[] a3 = merge(a1, a2);
System.out.println(Arrays.toString(a3));
// [12, 34, 56, 7, 8, 9, 10]
```

Array return answer

```
// Returns a new array containing all elements of a1
// followed by all elements of a2.
public static int[] merge(int[] a1, int[] a2) {
    int[] result = new int[a1.length + a2.length];

    for (int i = 0; i < a1.length; i++) {
        result[i] = a1[i];
    }
    for (int i = 0; i < a2.length; i++) {
        result[a1.length + i] = a2[i];
    }

    return result;
}
```