Building Java Programs

Chapter 7: Arrays

Lecture 7-2: Arrays as parameters and return values, text processing

Remember: why arrays?

- Storing a large amount of data
  - Printing the lines of a file in reverse order.
- Grouping related data
  - Tallying exam scores from 0 through 100.
- Accessing data multiple times, or in random order
  - Finding temps below the average of user-provided data.

Quick array initialization

<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

- Useful when you know what the array's elements will be.
- The compiler figures out the size by counting the values.

Traversals for printing

- Printing an array for debugging or final output:

```
int[] a = {2, 4, 6, 8};
System.out.print("(" + a[0]);
for (int i = 1; i < a.length; i++) {
    System.out.print(" " + a[i]);
}
System.out.println(")");
```

  - Output:

```
(2 4 6 8)
```

- **traversal**: An examination of each element of an array.
  - Traversal algorithms often take the following form:

```
for (int i = 0; i < <array>.length; i++) {
        do something with <array> [i];
}
```

Arrays.toString

- `Arrays.toString` accepts an array as a parameter and returns its data as a `String`, which you can print.

  - Example:

    ```
    int[] e = {0, 2, 4, 6, 8};
    e[1] = e[4] + e[3];
    System.out.println("e is " + Arrays.toString(e));
    ```

    Output:

    ```
    e is [0, 14, 4, 6, 8]
    ```

- Must `import java.util.*`

The Arrays class

- The `Arrays` class in package `java.util` has several useful static methods for manipulating arrays:

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

Arrays as parameters and return values

reading: 7.1

self-checks: #5, 8, 9

exercises: #1-10

Arrays as parameters

- Syntax (declaration):
  <method name>(***<type>***[] ***<parameter name>***)

  - Example:

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

Arrays as parameters

- Syntax (call):

  <method name>(***<array name>***);

  - Example:

```
public static void main(String[] args) {
    int[] iq = {126, 84, 149, 167, 95};
    double avg = average(iq);
    System.out.println("Average = " + avg);
}
```

Output:

```
Average = 124.2
```

Arrays passed by reference

- Arrays are objects.
  - Passed as parameters by *reference.*
    (Changes made in method also seen by caller.)

- 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[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] *= 2;
    }
}
```

  - Output:

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

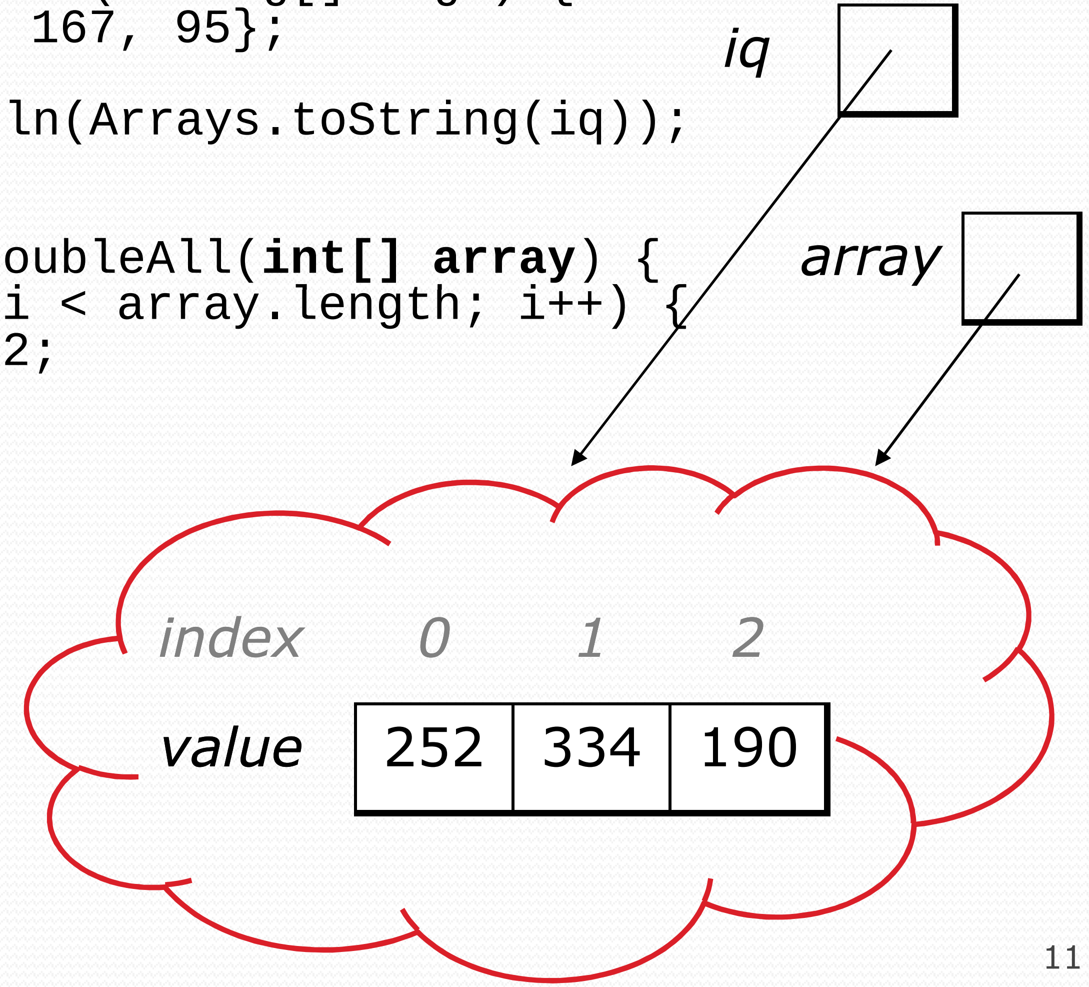
Array parameter diagram

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

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

- Output:

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



Arrays as return values

- Syntax (declaration):

```
public static <type>[] <method name>() {
```

  - Example:

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

Arrays as return values

- Syntax (call):

  <type>`[]` ***<name>*** `=` ***<method name>***`();`

  - 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]
```

Array parameter questions

- Write a method named `count` that accepts an array of integers and a target value and returns the number of times the value occurs.
- Write a method named `replace` that accepts an array of `int`s and two `int`s as parameters. The method should replace all occurrences of the first `int` with the second.
- Improve the previous Histogram program by making it use parameterized methods.

Array parameter answers

```
public static int count(int[] values, int target) {
    int count = 0;
    for (int i = 0; i < values.length; i++) {
        if (values[i] == target) {
            count++;
        }
    }
    return count;
}

public static void replace(int[] array, int val1, int val2) {
    for (int i = 0; i < array.length; i++) {
        if(array[i] == val1) {
            array[i] = val2;
        }
    }
}
```

Text processing

reading: 4.4

self-checks: #19-23

exercises: #5

Text processing

- **text processing**: Examining, editing, formatting text.
  - Often involves `for` loops to break up and examine a `String`
  - Examples:
    - Count the number of times 's' occurs in a file
    - Find which letter is most common in a file
    - Count A, C, T and Gs in `String`s representing DNA strands

Strings as arrays

- `String`s are represented internally as arrays.
  - Each character is stored as a value of primitive type `char`.
  - Strings use 0-based indexes, like arrays.
  - We can write algorithms to traverse `String`s.

  - Example:

```
String str = "Ali G.";
```

index	*0*	*1*	*2*	*3*	*4*	*5*
value	`'A'`	`'l'`	`'i'`	`' '`	`'G'`	`'.'`

Type char

- **char**: A primitive type representing a single character.
  - Literal char values are surrounded with apostrophe marks: 'a' or '4' or '\n' or '\''
  - You can have variables, parameters, returns of type char
  - You can compare char values with relational operators:
    - 'a' < 'b' and 'Q' != 'q'
    - An example that prints the alphabet:

```
for (char ch = 'a'; ch <= 'z'; ch++) {
    System.out.print(ch);
}
```

The charAt method

- Access a string's characters with its `charAt` method.

```
String word = console.next();
char firstLetter = word.charAt(0);
if (firstLetter == 'c') {
    System.out.println("That's good enough for me!");
}
```

- We can use for loops to examine each character.

```
String coolMajor = "CSE";
for (int i = 0; i < coolMajor.length(); i++) {
    System.out.println(coolMajor.charAt(i));
}
```

Output:

```
C
S
E
```

char vs. String

- `'h'` and `"h"` have different types

- char values are primitive; you can't call methods on them

```
char c = 'h';
c.toUpperCase();   // ERROR: "char cannot be dereferenced"
```

- Strings are objects; they contain methods

```
String s = "h";
int len = s.length();       // 1
char first = s.charAt(0);   // 'h'
```

Text processing question

- Write a method `tallyVotes` that accepts a `String` parameter and returns an array containing the number of McCain, Obama and independent voters.

```
// string stores votes: (M)cCain, (O)bama, (I)ndep.
String votes = "MOOOOOOMMMMMOOOOOOMOMMIMOMMIMOMMIIIIIIIIIIIII";
int[] tallies = tallyVotes(votes);
System.out.println(Arrays.toString(tallies));
```

  - Output:

```
[15, 15, 16]
```

Text processing answer

```
public static int[] tallyVotes(String votes) {
    int[] tallies = new int[3]; // M -> 0, O -> 1, I -> 2
    for(int i = 0; i < votes.length(); i++) {
        if(votes.charAt(i) == 'M') {
            tallies[0]++;
        } else if(votes.charAt(i) == 'O') {
            tallies[1]++;
        } else {    // votes.charAt(i) == 'I'
            tallies[2]++;
        }
    }
    return tallies;
}
```