

CSE 143

Lecture 1

Arrays (review)

slides created by Marty Stepp

<http://www.cs.washington.edu/143/>

Array declaration

type[] **name** = new **type**[**length**] ;

– Example:

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

– All elements' values are initially 0.

<i>index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>
<i>value</i>	0	0	0	0	0	0	0	0	0	0

Accessing elements

```
name[ index ]           // access  
name[ index ] = value; // modify
```

– Example:

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

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

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	27	0	0	-6	0	0	0	0	0	0

Out-of-bounds

- Legal indexes: between **0** and the **array's length - 1**.
 - Reading or writing any index outside this range will throw an `ArrayIndexOutOfBoundsException`.

- Example:

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

<i>index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>
<i>value</i>	0	0	0	0	0	0	0	0	0	0

The length field

name.length

- An array's length field stores its number of elements.

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

```
// output: 0 2 4 6 8 10 12 14
```

- It does not use parentheses like a String's `.length()`.

Quick initialization

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

– Example:

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

<i>index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>
<i>value</i>	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.

The Arrays class

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

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

Array as parameter

```
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);
```

Array as return

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

– Example:

```
public static int[] countDigits(int n) {  
    int[] counts = new int[10];  
    ...  
    return counts;  
}
```

• Call:

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

– Example:

```
int[] tally = countDigits(229231007);  
System.out.println(Arrays.toString(tally));
```

Exercise

- Write a method named `stutter` that accepts an array of integers as a parameter and returns a new array, twice as long as the original, with two copies of each original element.

- If the method were called in the following way:

```
int[] a = {4, 7, -2, 15, 6};  
int[] a2 = stutter(a);  
System.out.println(Arrays.toString(a2));
```

- The output produced would be:

```
[4, 4, 7, 7, -2, -2, 15, 15, 6, 6]
```

Exercise solutions

```
public static int[] stutter(int[] a) {
    int[] result = new int[a.length * 2];
    for (int i = 0; i < a.length; i++) {
        result[2 * i] = a[i];
        result[2 * i + 1] = a[i];
    }
    return result;
}
```

```
public static int[] stutter(int[] a) {
    int[] result = new int[a.length * 2];
    for (int i = 0; i < result.length; i++) {
        result[i] = a[i / 2];
    }
    return result;
}
```

Testing code

- Q: How can we tell if our `stutter` method works properly?
 - A: We must test it.
- Q: How do we test code?
 - A: Call the method several times and print/examine the results.
- Q: Can we test all possible usages of this method?
 - Q: Can we prove that the `stutter` code has no bugs?
 - A: No; exhaustive testing is impractical/impossible for most code.
 - A: No; testing finds bugs but cannot prove the absence of bugs.

How to test code

- **test case**: Running a piece of code once on a given input.
- Q: Which cases should we choose to test?
 - *equivalence classes of input*: Think about kinds of inputs:
 - positive vs. negative numbers vs. 0; `null` (maybe)
 - unique values vs. duplicates (consecutive and non-consecutive)
 - an empty array; a 1-element array; a many-element array
- Q: What are some properties to look for in testing code?
 - *boundaries*: Hits cases close to a relevant boundary, e.g. the maximum allowed value, the first/last element in an array, etc.
 - *code coverage*: Hits all paths through code (`if/elses`, etc.)
 - *preconditions*: What does the method assume? Does the code ever violate those assumptions?

Exercise

- Write a short piece of code that tests the `stutter` method.
 - Decide on a group of test input cases.
 - For each test case:
 - Print the array's contents before and after stuttering.
 - Print whether the test was successful or failed.

Exercise solution 1

```
public static void main(String[] args) {  
    int[] a1 = {1, 2, 4, 5, 6};  
    int[] a2 = stutter(a1);  
    System.out.println(Arrays.toString(a2));  
    ...  
}
```

- Pros:
 - simple, short
- Cons:
 - must manually check output to see if it is correct
 - must copy/paste to create each test case (redundant)

Exercise solution 2

```
public static void main(String[] args) {
    test(new int[] {1, 2, 4, 5, 6, 8},
          new int[] {1, 1, 2, 2, 4, 4, 5, 5, 6, 6, 8, 8});
    test(new int[] {0, 0, 7, 9},
          new int[] {0, 0, 0, 0, 7, 7, 9, 9});
    test(new int[] {-50, 95, -9876},
          new int[] {-50, -50, 95, 95, -9876, -9876});
    test(new int[] {42}, new int[] {42, 42});
    test(new int[] {}, new int[] {});
}
```

```
public static void test(int[] a, int[] expected) {
    int[] a2 = stutter(a);
    System.out.print(Arrays.toString(a) + " -> " +
                     Arrays.toString(a2) + " : ");
    if (Arrays.equals(a2, expected)) {
        System.out.println("Pass");
    } else {
        System.out.println("FAIL!!!");
    }
}
```