

# Programming Languages and Techniques (CIS120)

## Lecture 21

Mar 2, 2012

Java Programming: Static Methods, Arrays

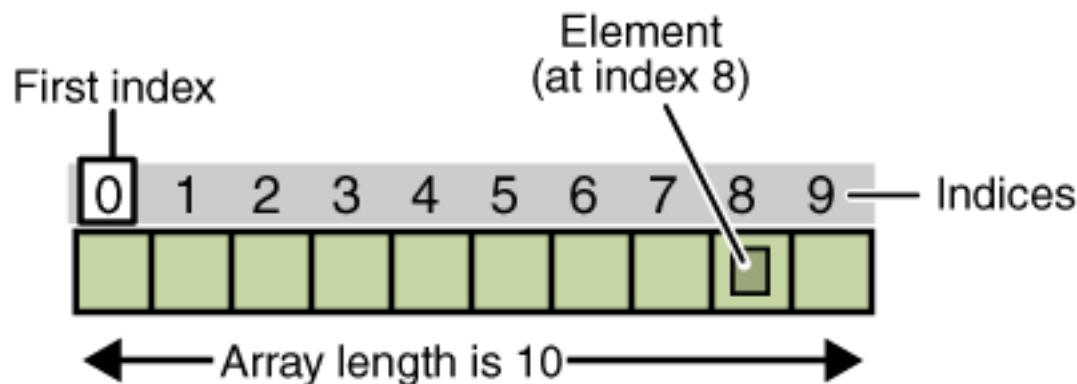
# Announcements

- HW06 Grace period until 11:59:59pm tonight
- HW07 is available on the web
  - Image processing in Java
  - Due next Thursday, March 15<sup>th</sup> at 11:59:59pm
- Have a good break!

# Java arrays

# Java Arrays: Indexing

- An array is a sequentially ordered collection of values that can be indexed in *constant* time.
- Index elements from 0



- Basic array expression forms

`a[i]` access element of array `a` at index `i`

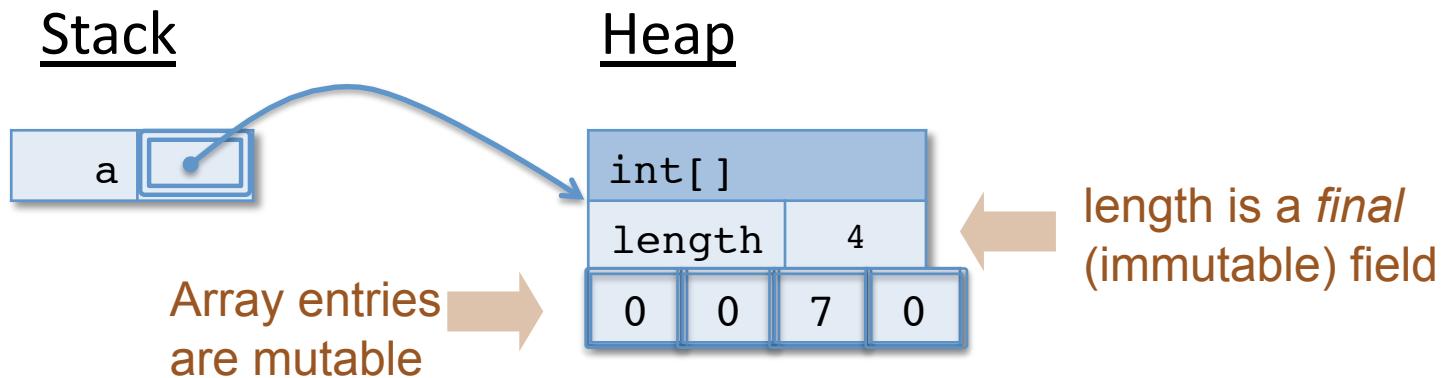
`a[i] = e` assign `e` to element of array `a` at index `i`

`a.length` get the number of elements in `a`

# Java Arrays: Dynamic Creation

- Create an array `a` of size `n` with elements of type `C`  
`C[ ] a = new C[n];`
- Arrays are objects that live in the heap, values with array type are mutable references

```
int[ ] a = new int[ 4 ];  
a[ 2 ] = 7;
```



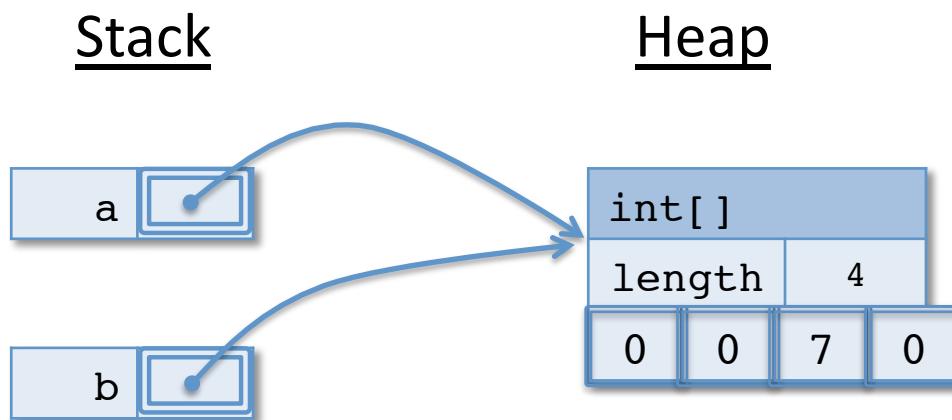
# Java Arrays: Static Initialization

```
int[ ] myArray = { 100, 200, 300, 400, 500,  
                   600, 700, 800, 900, 1000};  
  
String[ ] yourArray = { "foo", "bar", "baz" };  
  
Point[ ] herArray = { new Point(1,3),  
                     new Point(5,4) };  
  
herArray = new Point[ ] { new Point(2,3),  
                         new Point(6,5) };
```

# Java Arrays: Aliasing

- Variables of array type are references and can be aliases

```
int[ ] a = new int[4];
int[ ] b = a;
a[2] = 7;
System.out.println(b[2]);
```



# Array Iteration

# For loops

initialization      loop condition      update

The diagram illustrates the structure of a for loop. It features three orange arrows pointing downwards from the labels 'initialization', 'loop condition', and 'update' to specific parts of the code. The 'initialization' arrow points to the assignment in the first iteration line. The 'loop condition' arrow points to the comparison in the same line. The 'update' arrow points to the increment in the same line. Below these, a large orange box encloses the entire loop body, which contains the assignment to 'total' and the closing brace '}'.

```
for (int i = 0; i < a.length; i++) {  
    total += a[i];  
}
```

```
static double sum(double[ ] a) {  
    double total = 0;  
    for (int i = 0; i < a.length; i++) {  
        total += a[i];  
    }  
    return total;  
}
```

# Multi-Dimensional Arrays

A 2-d array is just an array of arrays...

```
String[][] names = {{"Mr. ", "Mrs. ", "Ms. "},  
                    {"Smith", "Jones"}};  
  
System.out.println(names[0][0] + names[1][0]);  
    // --> Mr. Smith  
System.out.println(names[0][2] + names[1][1]);  
    // --> Ms. Jones
```

`String[][]` just means `(String[])[ ]`  
`names[1][1]` just means `(names[1])[1]`  
More brackets → more dimensions

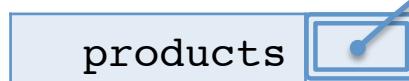
# Multi-Dimensional Arrays

```
int[][] products = new int[5][];
for(int col = 0; col < 5; col++) {
    products[col] = new int[col+1];
    for(int row = 0; row <= col; row++) {
        products[col][row] = col * row;
    }
}
```

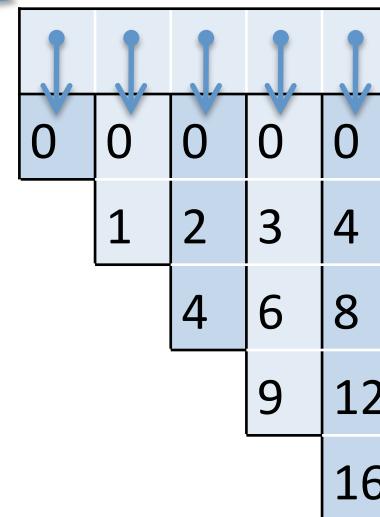
# Multi-Dimensional Arrays

```
int[][] products = new int[5][];
for(int col = 0; col < 5; col++) {
    products[col] = new int[col+1];
    for(int row = 0; row <= col; row++) {
        products[col][row] = col * row;
    }
}
```

Stack



Heap



Note: This heap picture is simplified – it omits the class identifiers and length fields for all 6 of the arrays depicted.

(Contrast with the array shown earlier.)

# Demo

ArrayExamples.java