Programming Languages and Techniques (CIS120)

Lecture 23

Static Methods, Java Arrays

Chapters 20 & 21

Announcements

- HW6: Java Programming (Pennstagram)
  - Tuesday, November 5 at 11:59:59pm
- Reminder: please complete mid-semester survey
  - See post on Piazza
- Upcoming: Midterm 2
  - Friday, November 8th in class
  - Coverage: mutable state, queues, deques, GUI, Java

Static Methods and Fields

functions and global state

Java Main Entry Point

```
class MainClass {

    public static void main (String[] args) {
          …
    }

}
```

- Program starts running at `main`
  - `args` is an array of `Strings` (passed in from the command line)
  - must be public
  - returns `void` (i.e. is a command)

- What does *static* mean?

How familiar are you with the idea of Java's "static" methods and fields?

- I haven’t heard of the idea of “static”.
- I’ve used “static” without knowing what it means.
- I have some familiarity with the difference between “static” and “dynamic”.
- I totally get it.

Static method example

```
public class Max {

  public static int max (int x, int y) {
    if (x > y) {
      return x;
    } else {
      return y;
    }
  }

  public static int max3(int x, int y, int z) {
     return max(max(x,y), z);
  }
}
```

closest analogue of top-level functions in OCaml, but must be a member of some class

Internally (within the same class), call with just the method name

```
public class Main {

 public static void main (String[] args) {

       System.out.println(Max.max(3,4));
       return;
    }
}
```

main method must be static; it is invoked to start the program running

Externally, prefix with name of the class

mantra

Static == Decided at *Compile* Time
Dynamic == Decided at *Run* Time

Static vs. Dynamic Methods

- Static Methods are *independent* of object values
  - They are associated with a whole Class not an instance of the Class
  - Similar to OCaml functions
  - Cannot refer to the local state of objects (fields or normal methods)
- Use static methods for:
  - Non-OO programming
  - Programming with primitive types: Math.sin(60), Integer.toString(3), Boolean.valueOf("true")
  - "`public static void main`"
- "Normal" methods are *dynamic*
  - Need access to the local state of the particular object on which they are invoked
  - We know only at *runtime* which method will get called

```
void moveTwice (Displaceable o) {
   o.move (1,1); o.move(1,1);
}
```

Method call examples

- Calling a (dynamic) method of an object (o) that returns a number:

```
x = o.m() + 5;
```

- Calling a static method of a class (C) that returns a number:

```
x = C.m() + 5;
```

- Calling a method that returns void:

Static

```
C.m();
```

Dynamic

```
o.m();
```

- Calling a static or dynamic method in a method of the same class:

Either

```
m();
```

Static

```
C.m();
```

Dynamic

```
this.m();
```

- Calling (dynamic) methods that return objects:

```
x = o.m().n();
x = o.m().n().x().y().z().a().b().c().d().e();
```

Which staic methoc can we add to this class?

```
public class Counter {

  private int r;

  public Counter () {
    r = 0;
  }

  public int inc () {
    r = r + 1;
    return r;
  }

  // A,B, or C here ?

}
```

```
public static int dec () {
   r = r - 1;
   return r;
}
```

```
public static int inc2 () {
   inc();
   return inc();
}
```

```
public static int getInitialVal () {
   return 0;
}
```

Which **static** method can we add to this class?

```
public class Counter {

  private int r;

  public Counter () {
    r = 0;
  }

  public int inc () {
    r = r + 1;
    return r;
  }

  // A,B, or C here ?

}
```

A.

```
public static int dec () {
   r = r - 1;
   return r;
}
```

B.

```
public static int inc2 () {
   inc();
   return inc();
}
```

C.

```
public static int getInitialVal () {
   return 0;
}
```

Answer: C

Static vs. Dynamic Class Members

```
public class FancyCounter {
  private int c = 0;
  private static int total = 0;

  public int inc () {
    c += 1;
    total += 1;
    return c;
  }

  public static int getTotal () {
    return total;
  }
}
```

```
FancyCounter c1 = new FancyCounter();
FancyCounter c2 = new FancyCounter();
int v1 = c1.inc();
int v2 = c2.inc();
int v3 = c1.getTotal();
System.out.println(v1 + " " + v2 + " " + v3);
```

Static Class Members

- Static methods can depend *only* on other static things
  - Static fields and methods, from the same or other classes
- Static methods *can* create *new* objects and use them
  - This is typically how `main` works
- `public static` fields are the "global" state of the program
  - Mutable global state should generally be avoided
  - Immutable global fields are useful: for constants like pi

```
public static final double PI = 3.14159265359793238462643383279;
```

Style: naming conventions

Kind	Part-of-speech	Example
class	noun	RacingCar
field / variable	noun	initialSpeed
static final field (constants)	noun	MILES_PER_GALLON
method	verb	shiftGear

- Identifiers consist of alphanumeric characters and _ and cannot start with a digit
- The larger the scope, the more *informative* the name should be
- Conventions are important: variables, methods and classes can have the same name

Why naming conventions matter

```
public class Turtle {
  private Turtle Turtle;
  public Turtle() { }

  public Turtle Turtle (Turtle Turtle) {
    return Turtle;
  }
}
```

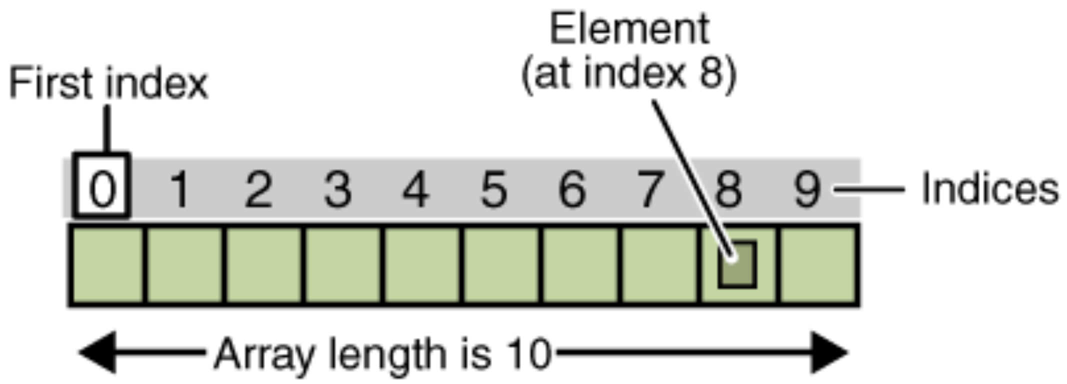
Many more details on good Java style here:
http://www.seas.upenn.edu/~cis120/current/java_style.shtml

Java arrays

Working with static methods

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

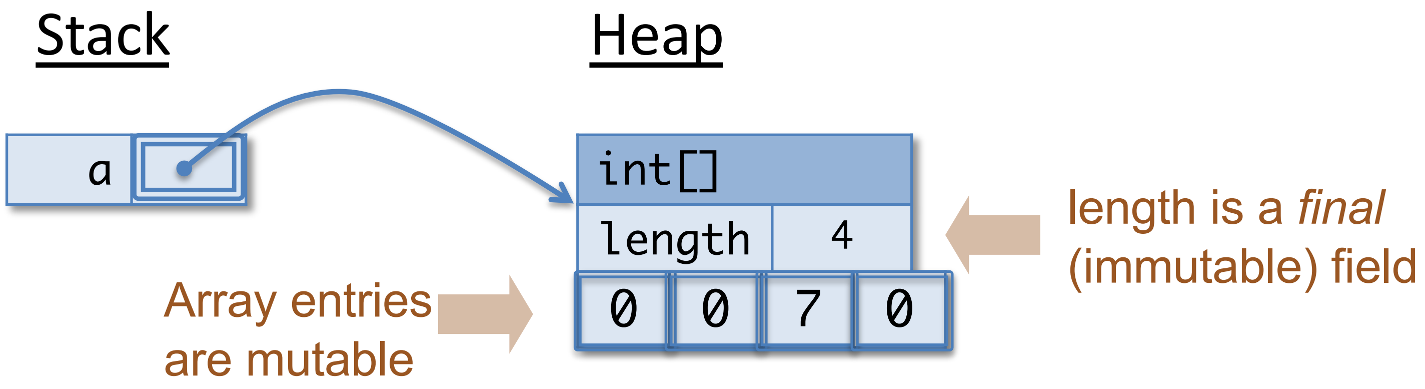
```
    C[] a = new C[n];
```

- Create an array of four integers, initialized as given:

```
    int[] x = {1, 2, 3, 4};
```

- Arrays live in the heap; values with array type are mutable references:

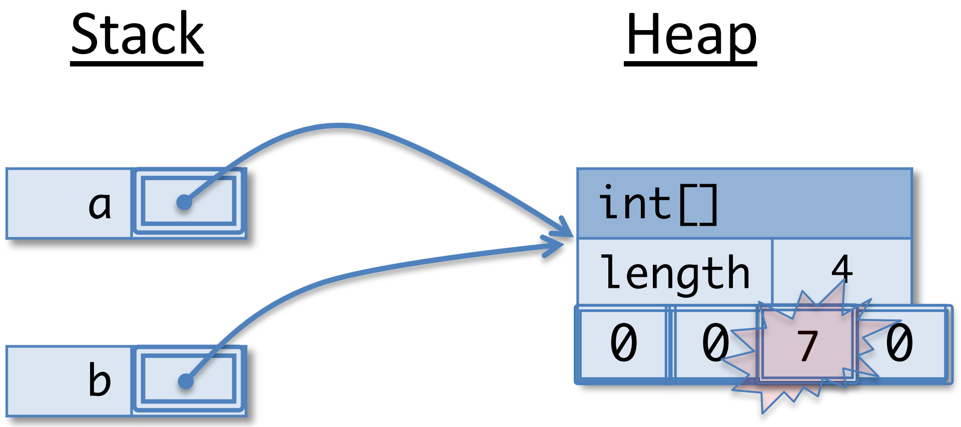
```
int[] a = new int[4];
a[2] = 7;
```


Stack
Heap
a
int[]
length
4
0
0
7
0
length is a *final* (immutable) field
Array entries are mutable

Java Arrays: Aliasing

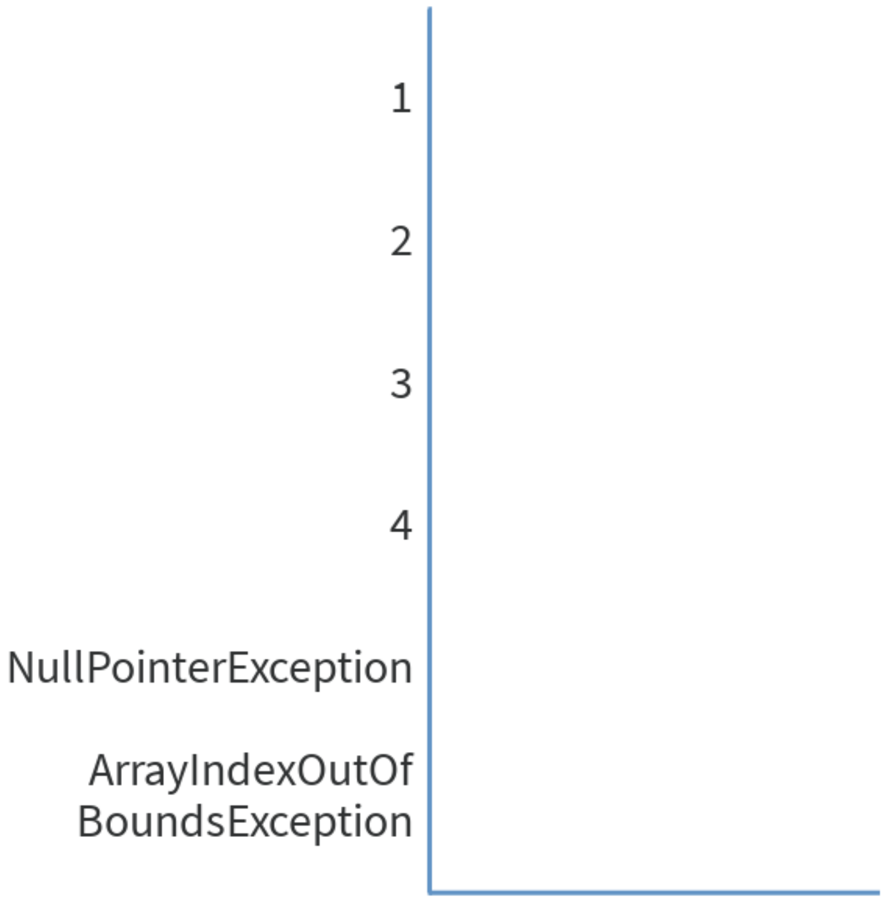
- Variables of array type are references and can be aliases

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



What is the value of *ans* at the end of this program?

```java
int[] a = {1, 2, 3, 4};
int ans = a[a.length];
```



What is the value of ans at the end of this program?

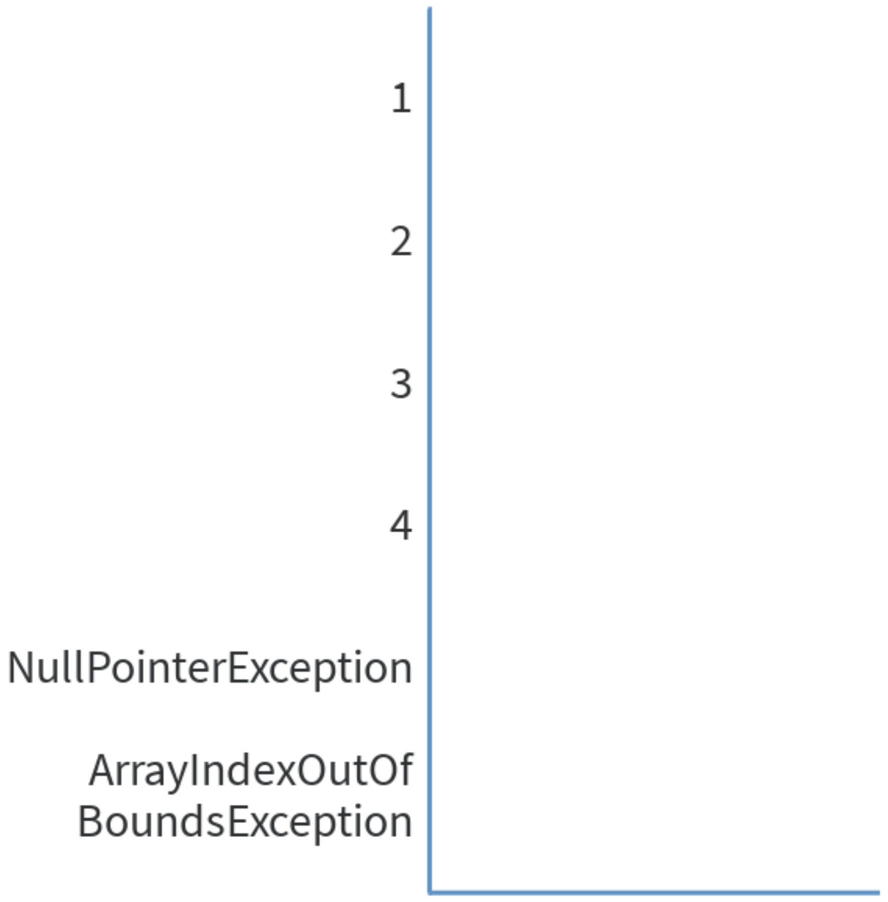
```
int[] a = {1, 2, 3, 4};
int ans = a[a.length];
```

1. 1
2. 2
3. 3
4. 4
5. NullPointerException
6. ArrayIndexOutOfBoundsException

Answer: ArrayIndexOutOfBoundsException

What is the value of *ans* at the end of this program?

```java
int[] a = null;
int ans = a.length;
```



What is the value of ans at the end of this program?

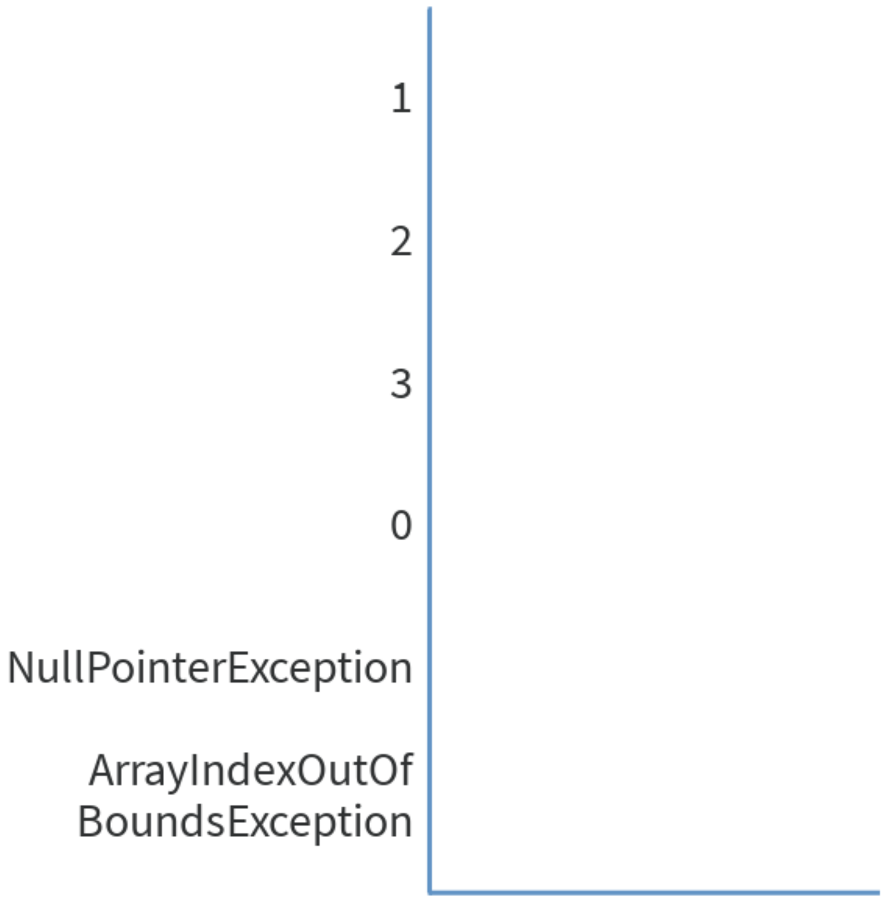
```java
int[] a = null;
int ans = a.length;
```

1. 1
2. 2
3. 3
4. 0
5. NullPointerException
6. ArrayIndexOutOfBoundsException

Answer: NullPointerException

What is the value of *ans* at the end of this program?

```
int[] a = {};
int ans = a.length;
```



What is the value of ans at the end of this program?

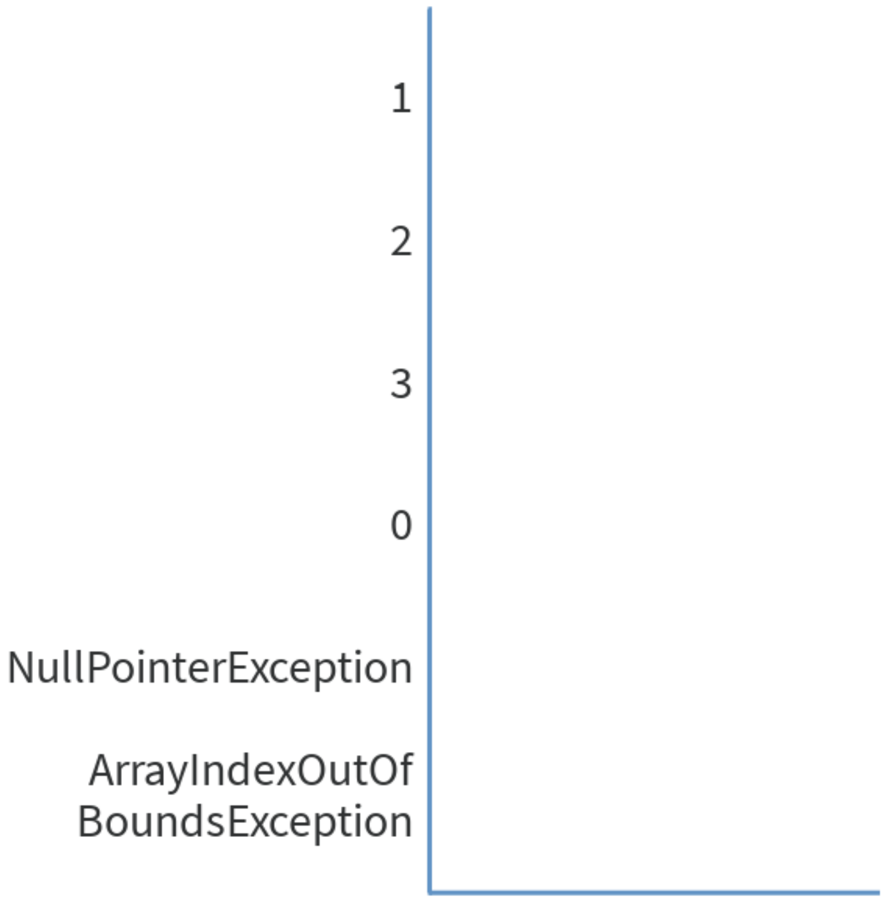
```
int[] a = {};
int ans = a.length;
```

1. 1
2. 2
3. 3
4. 0
5. NullPointerException
6. ArrayIndexOutOfBoundsException

Answer: 0

What is the value of *ans* at the end of this program?

```java
int[] a = {1, 2, 3, 4};
int[] b = a;
b[0] = 0;
int ans = a[0];
```



What is the value of ans at the end of this program?

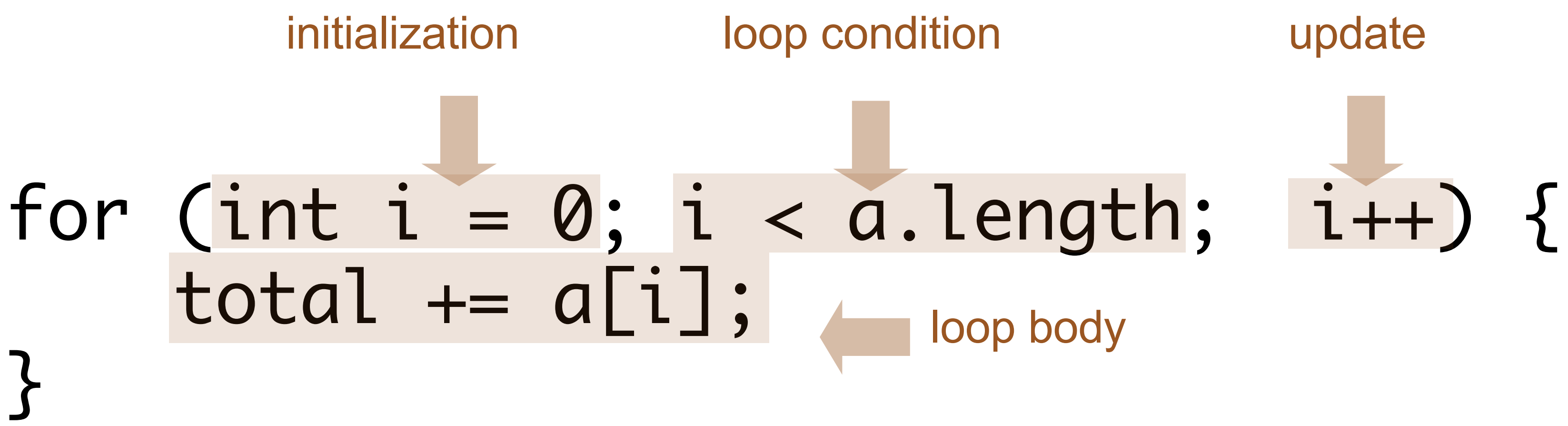
```
int[] a = {1, 2, 3, 4};
int[] b = a;
b[0] = 0;
int ans = a[0];
```

1. 1
2. 2
3. 3
4. 0
5. NullPointerException
6. ArrayIndexOutOfBoundsException

Answer: 0

Array Iteration

For loops



```
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;
}
```

General pattern for computing info about an array

Multidimensional Arrays

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

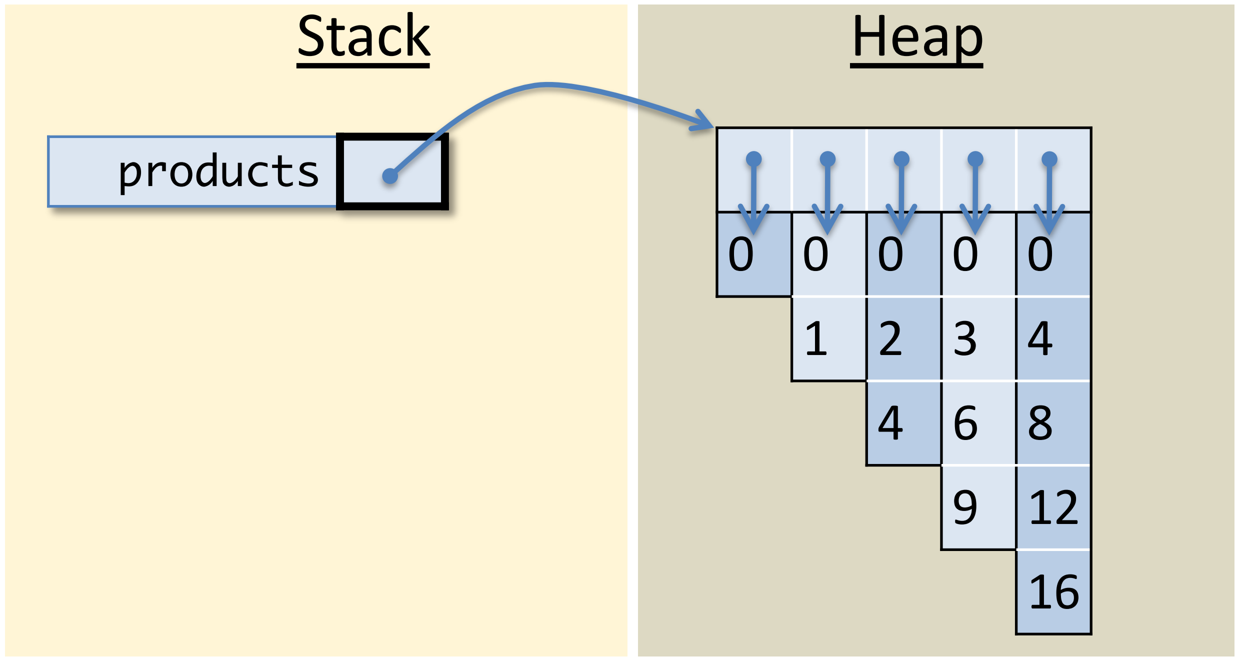
Multi-Dimensional Arrays

```
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;
   }
}
```

What would a “Java ASM” stack and heap look like after running this program?

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;
   }
}
```



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.)

Note also that orientation doesn't matter on the heap.

Demo

ArrayDemo.java

Design Exercise: Resizable Arrays

Arrays that grow without bound.

Resizable Arrays

```java
public class ResArray {

  /** Constructor, takes no arguments. */
  public ResArray() { … }

  /** Access the array at position i. If position i has not yet
   * been initialized, return 0.
   */
  public int get(int i) { … }

  /** Modify the array at position i to contain the value v. */
  public void set(int i, int v) { … }

  /** Return the extent of the array. */
  public int getExtent() { … }

}
```

Object Invariant: extent is 1 past the last nonzero value in data (can be 0 if the array is all zeros)