Arrays, Java ASM
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];
```

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 = 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?

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

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

Answer: 0
Announcements

• HW6: Java Programming (Pennstagram)
  – Due: next Tuesday at 11:59pm
  – Note: Run JUnit tests (under Tools menu)

• Reminder: please complete mid-semester survey
  – See post on Piazza

• Upcoming: Midterm 2
  – Friday, March 23rd in class
  – Coverage: mutable state, queues, deques, GUI, Java
The Java Abstract Stack Machine

Objects, Arrays, and Static Methods
Java Abstract Stack Machine

• Similar to OCaml Abstract Stack Machine
  – Workspace
    • Contains the currently executing code
  – Stack
    • Remembers the values of local variables and "what to do next" after function/method calls
  – Heap
    • Stores reference types: objects and arrays

• Key differences:
  – Everything, including stack slots, is mutable by default
  – Objects store *what class was used to create them*
  – *Arrays store type information and length*
  – *New component: Class table (coming soon)*
Heap Reference Values

Arrays
- Type of values that it stores
- Length
- Values for all of the array elements

```java
int[] a = {0, 0, 7, 0};
```

Objects
- Name of the class that constructed it
- Values for all of the fields

```java
class Node {
    private int elt;
    private Node next;
    ...
}
```

- Fields may or may not be mutable
- Public/private not tracked by ASM
Multidimensional Arrays
Multi-Dimensional Arrays

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

```java
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
```

`names[1][1]` just means `(names[1])[1]`

More brackets → more dimensions
```java
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?
`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 last time.)  

Note also that orientation doesn’t matter on the heap.
More Array Examples

See ArrayExamples.java
Objects on the ASM
public class Node {
    public int elt;
    public Node next;
    public Node(int e0, Node n0) {
        elt = e0;
        next = n0;
    }
}

public class Test {
    public static void main(String[] args) {
        Node n1 = new Node(1, null);
        Node n2 = new Node(2, n1);
        Node n3 = n2;
        n3.next.next = n2;
        Node n4 = new Node(4, n1.next);
        n2.next.elt = 9;
        System.out.println(n1.elt);
        Answer: 9
    }
}
```java
Node n1 = new Node(1, null);
Node n2 = new Node(2, n1);
Node n3 = n2;
n3.next.next = n2;
Node n4 = new Node(4, n1.next);
n2.next.elt = 9;
```
Node n1 = ;
Node n2 = new Node(2,n1);
Node n3 = n2;
n3.next.next = n2;
Node n4 = new Node(4,n1.next);
n2.next.elt = 9;

Note: we’re skipping details here about how the constructor works. We’ll fill them in next week. For now, assume the constructor allocates and initializes the object in one step.
Workspace

Node n2 = new Node(2,n1);
Node n3 = n2;
n3.next.next = n2;
Node n4 = new Node(4,n1.next);
n2.next.elt = 9;
Node n2 = null;
Node n3 = n2;
n3.next.next = n2;
Node n4 = new Node(4,n1.next);
n2.next.elt = 9;
Workspace

Node n3 = n2;
n3.next.next = n2;
Node n4 = new Node(4, n1.next);
n2.next.elt = 9;
Workspace

n3.next.next = n2;
Node n4 = new Node(4,n1.next);
n2.next.elt = 9;
n3.next.next = n2;
Node n4 = new Node(4, n1.next);
n2.next.elt = 9;
n3.next.next = n2;
Node n4 = new Node(4,n1.next);
n2.next.elt = 9;
Node n4 = new Node(4, n1.next);
n2.next.elt = 9;
```java
Node n4 = null;
n2.next.elt = 9;
```
n2.next.elt = 9;
n2.next.elt = 9;
n2.next.elt = 9;
What is the value of `ans` at the end of this program?

```java
Counter[] a = { new Counter(), new Counter() };
Counter[] b = a;
a[0].inc();
b[0].inc();
int ans = a[0].inc();
```

1. 1
2. 2
3. 3
4. 0
5. NullPointerException
6. ArrayIndexOutOfBoundsException
What is the value of ans at the end of this program?

```
public class Counter {
    private int r;
    public Counter () {
        r = 0;
    }
    public int inc () {
        r = r + 1;
        return r;
    }
}
```

Counter[] a = { new Counter(), new Counter() };
Counter[] b = { a[0], a[1] };
a[0].inc();
b[0].inc();
int ans = a[0].inc();
What does the ASM look like at the end of this program?

```java
Counter[] a = { new Counter(), new Counter() };
Counter[] b = { a[0], a[1] };
a[0].inc();
b[0].inc();
int ans = a[0].inc();

public class Counter {
    private int r;

    public Counter () {
        r = 0;
    }

    public int inc () {
        r = r + 1;
        return r;
    }
}
```
What does the ASM look like at the end of this program?

```java
Counter[] a = { new Counter(), new Counter() };
Counter[] b = { a[0], a[1] };
a[0].inc();
b[0].inc();
int ans = a[0].inc();
```

```java
public class Counter {
    private int r;
    public Counter () {
        r = 0;
    }
    public int inc () {
        r = r + 1;
        return r;
    }
}
```
Design Exercise: Resizable Arrays

Arrays that grow without bound.
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 always 1 past the last nonzero value in data
(or 0 if the array is all zeros)
private int extent = 0;
    /* INVARIANT: extent = 1+index of last nonzero *
     * element, or 0 if all elements are 0. */

/** Modify the array at position i to contain the value v. */
public void set(int idx, int val) {
    if (idx < 0) {
        throw new IllegalArgumentException();
    }
    grow(idx);
    data[idx] = val;
    if (val != 0 && idx+1 > extent) {
        extent = idx+1;
    }
    if (val == 0 && idx+1 == extent) {
        while (extent > 0 && data[extent-1] == 0) {
            extent--;
        }
    }
}

/** Return the extent of the array. */
public int getExtent() {
    return extent;
}
ResArray ASM

Workspace

```
ResArray x = new ResArray();
x.set(3,2);
x.set(4,1);
x.set(4,0);
```

Stack

Heap
ResArray x = new ResArray();
x.set(3,2);
x.set(4,1);
x.set(4,0);
ResArray ASM

Workspace

```java
ResArray x = new ResArray();
x.set(3, 2);
x.set(4, 1);
x.set(4, 0);
```

Stack

Heap

<table>
<thead>
<tr>
<th>ResArray</th>
<th>int[]</th>
<th>int[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>length 0</td>
<td>4</td>
</tr>
<tr>
<td>extent</td>
<td>length 4</td>
<td>0 0 0 2</td>
</tr>
</tbody>
</table>
ResArray ASM

Workspace

```
ResArray x = new ResArray();
x.set(3, 2);
x.set(4, 1);
x.set(4, 0);
```

Stack

Heap

<table>
<thead>
<tr>
<th>ResArray</th>
<th>data</th>
<th>extent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int[]</th>
<th>length</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
ResArray x = new ResArray();
x.set(3, 2);
x.set(4, 1);
x.set(4, 0);
ResArray x = new ResArray();
x.set(3, 2);
x.set(4, 1);
x.set(4, 0);
ResArray x = new ResArray();
x.set(3, 2);
x.set(4, 1);
x.set(4, 0);
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() { ... }

    /** The smallest prefix of the ResArray
     * that contains all of the nonzero values, as a normal array.
     */
    public int[] values() { ... }
}
Values Method

```java
public int[] values() {
    int[] values = new int[extent];
    for (int i=0; i<extent; i++) {
        values[i] = data[i];
    }
    return values;
}
```

Or maybe we can do it more straightforwardly? ...

```java
public int[] values() {
    return data;
}
```
ResArray x = new ResArray();
x.set(3, 2);
int[] y = x.values();
y[3] = 0;
ResArray ASM

Workspace

```java
ResArray x = new ResArray();
x.set(3, 2);
int[] y = x.values();
y[3] = 0;
```

Stack  Heap

<table>
<thead>
<tr>
<th>x</th>
<th>ResArray</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>data</td>
</tr>
<tr>
<td></td>
<td>extent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
ResArray ASM

Workspace

ResArray x = new ResArray();
x.set(3, 2);
int[] y = x.values();
y[3] = 0;

Stack

Heap

ResArray

<table>
<thead>
<tr>
<th>data</th>
<th>extent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

int[]

<table>
<thead>
<tr>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

invariant violation!
Object encapsulation

- All modification to the state of the object must be done using the object's own methods.

- Use encapsulation to preserve invariants about the state of the object.

- Enforce encapsulation by not returning aliases from methods.