Programming Languages and Techniques (CIS120)

Lecture 24
March 18, 2016

The Java ASM
What is the value of ans at the end of this program?

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

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

• Midterm 2, Tuesday evening, 6:15-8:15 PM
  – Topics on Monday's slides
  – Practice exams on website
  – Sign up for makeup exam ASAP (if necessary)
  – Exam location by last name (same as last time)
    A - Schwartz: MEYH B1
    Shah - Z: DRLB A8

• Review session
  – Sunday March 20th
  – Towne 100
  – 6-8PM
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

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

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

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

- Length never mutable; elements always mutable
- Fields may or may not be mutable public/private not tracked by ASM
What does the heap look like at the end of this program?

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;
  }
}
```
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
```

String[][] just means (String[])[]
names[1][1] just means (names[1])[1]
More brackets → more dimensions
```
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 does the stack and heap look like on the ASM after running this program?
```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;
    }
}
```

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.
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
    }
}
### Workspace

```
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 = 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;

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.
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;
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;
Workspace

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

Stack

n1
n2
n3

Heap

Node
elt 1
next null

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

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

```java
n2.next.elt = 9;
```
n2.next.elt = 9;
Workspace

n2.next.elt = 9;

Stack

<table>
<thead>
<tr>
<th>n1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n2</td>
<td></td>
</tr>
<tr>
<td>n3</td>
<td></td>
</tr>
<tr>
<td>n4</td>
<td></td>
</tr>
</tbody>
</table>

Heap

Node

<table>
<thead>
<tr>
<th>elt</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td></td>
</tr>
</tbody>
</table>

Node

<table>
<thead>
<tr>
<th>elt</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td></td>
</tr>
</tbody>
</table>

Node

<table>
<thead>
<tr>
<th>elt</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td></td>
</tr>
</tbody>
</table>
Array Examples Demo