Programming Languages and Techniques (CIS120)

Lecture 23
Oct 26, 2015
Resizable Arrays Demo
Java ASM
Announcements

• HW 06 due Thursday at midnight

• Midterm II, in class, a week from Friday (Nov 6th)

• Java Bootcamp
  – TONIGHT 6:00-8:00pm
  – Levine 100 (Wu & Chen)
  – Pizza!
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
Design Exercise: Resizable Arrays

Arrays that grow without bound.
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
  – Heap objects store *dynamic class information*
Heap Values

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

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

Arrays
• Type of values that it stores
• Length
• Values for all of the fields

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

Node
<table>
<thead>
<tr>
<th>elt</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>null</td>
</tr>
</tbody>
</table>

fields may or may not be mutable

int[]
<table>
<thead>
<tr>
<th>length</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

length never mutable; elements always mutable
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() { … }

}
Workspace

```java
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

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

Stack
Heap

- Stack: `x`
- Heap: `ResArray` with:
  - `data`: empty
  - `extent`: 4
- Heap: `int[]` with:
  - `length`: 4
  - `0 0 0 2`
ResArray ASM

Workspace

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

Stack
Heap

ResArray

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

int[]

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

Workspace

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>data</th>
<th>extent</th>
</tr>
</thead>
</table>

Int[]

<table>
<thead>
<tr>
<th>length</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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() { ... }
}
public int[] values() {
    int[] values = new int[extent];
    for (int i=0; i<extent; i++) {
        values[i] = data[i];
    }
    return values;
}

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

resArray 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.