Announcements

• HW 07 due Monday at midnight

• Exam II, in class, two weeks from today
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

• 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

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

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

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

<table>
<thead>
<tr>
<th>Node</th>
<th>elt</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>null</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>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Fields may or may not be mutable

Length never mutable
Elements always mutable
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 always 1 past the last nonzero value in data (or 0 if the array is all zeros)
ResArray ASM

Workspace

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

Stack	Heap
ResArray ASM

Workspace

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

Stack

```
x
```

Heap

```
ResArray
<table>
<thead>
<tr>
<th>data</th>
<th>extent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

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

ResArray

data

extent 4

int[]

length 0

int[]

length 4

0 0 0 2
ResArray ASM

Workspace

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

Stack	Heap

```
x
ResArray
\- data
| extent | 4 |
int[]
| length | 4 |
| 0      | 0 |
| 0      | 0 |
| 0      | 2 |
```
ResArray ASM

Workspace

```java
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>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
ResArray ASM

Workspace

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

Stack

```
ResArray
```

Heap

```
data
extent
```

```
int[]
<table>
<thead>
<tr>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
```

```text
0 0 0 2 0 0 0 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() { ... }
}

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

Workspace

ResArray data
extent

Stack
Heap

int[]
length

x

ResArray

y

CIS120 / Spring 2013
```java
ResArray x = new ResArray();
x.set(3, 2);
int[] y = x.values();
y[3] = 0;
```

Workspace

- **ResArray x**
  - data: [0, 0, 0, 0]
  - extent: 4

- **int[] y**
  - values: [0, 0, 0, 0]

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.
Object Aliasing example

```java
public class Node {
    public int elt;
    public Node next;
    public Node(int e0, Node n0) {
        elt = e0;
        next = n0;
    }
}

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 = 17;
    System.out.println(n1.next);
}
```
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 = 17;

Stack

Heap
Constructing an Object

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 = 17;

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 = 17;
Node n2 = ;
Node n3 = n2;
n3.next.next = n2;
Node n4 = new Node(4,n1.next);
n2.next.elt = 17;
Constructing an Object

Workspace

Node n3 = n2;
n3.next.next = n2;
Node n4 = new Node(4, n1.next);
n2.next.elt = 17;

Stack

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 = 17;
ConstrucSng an Object

Workspace

n3.next.next = n2;
Node n4 = new Node(4,n1.next);
n2.next.elt = 17;

Stack

Heap

Node
elt 1
next null

Node
elt 2
next null

Node
elt 2
next null

Node
elt 1
next null

Node
elt 2
next null
Workspace

n3.next.next = n2;
Node n4 = new Node(4,n1.next);
n2.next.elt = 17;

Stack

Heap

Node

elt 1
next

Node

elt 2
next

Node

workspace

Node n4 = new Node(4,n1.next);
n2.next.elt = 17;

stack

heap

Node

elt

next

Node

elt

next

Node

elt

next

Node

elt

next
Node n4 = null;
n2.next.elt = 17;
n2.next.elt = 17;
Constructing an Object

Workspace

n2.next.el = 17;

Stack

Heap

Node
elt 1
next

Node
elt 2
next

Node
elt 4
next
Constructing an Object

Workspace

n2.next.elt = 17;

Stack

Heap

Node
elt  17
next

Node
elt  2
next

Node
elt  4
next
Types and Subtyping
Why Static Types?

• Types stop you from using values incorrectly
  – 3.m()
  – if (3) { return 1; } else { return 2; }
  – 3 + true
  – (new Counter()).m()

• All expressions have types
  – 3 + 4 has type int
  – “A”.toLowerCase() has type String
  – new ResArray() has type ResArray

• How do we know if x.m() is correct? or x+3?
  – depends on the type of x
  – variable declarations specify types of variables

• Type restrictions preserve the types of variables
  – assignment "x = v" must be to values with compatible types
  – methods "o.m(3)" must be called with compatible argument types

• HOWEVER: in Java, values can have multiple types....
Subtyping

- **Definition**: Type A is a *subtype* of type B if A can do anything that B can do. Type B is called the *supertype* of A.

- **Example**: A class that implements an interface is a subtype of the interface

```java
interface Area {
    public double getArea ();
}

public class Circle implements Area {
    private double r;
    private Point p;
    public Circle (double x0, double y0, double r0) {
        r = r0; p = new Point(x0,y0);
    }
    public double getArea () {
        return 3.14159 * r * r;
    }
    public double getRadius () { return r; }
}
```
Subtyping and Variables

- A variable declared with type A can store any object that is a subtype of A

  ```java
  Area a = new Circle(1, new Point(2,3));
  ```

  supertype of Circle  subtype of Area

- Methods with parameters of type A must be called with arguments that are subtypes of A

  ```java
  static void double m (Area x) {
      return x.getArea() * 2;
  }
  ...
  C.m( new Circle(1, new Point(2,3)) );
  ```
Subtypes and Supertypes

- An interface represents a *point of view* about an object
- Classes can implement *multiple* interfaces

Types can have many different supertypes / subtypes
"Static" types vs. "Dynamic" classes

• The **static type** of an *expression* is a type that describes what we (and the compiler) know about the expression at compile-time (without thinking about the execution of the program)
  
  Displaceable `x;`

• The **dynamic class** of an *object* is the class that it was constructed from at run time

  ```
  x = new Point(1,2)
  ```

• In OCaml, we only had static types

• In Java, we also have dynamic classes because of objects
  – The dynamic class will always be a *subtype* of its static type
Static type vs. Dynamic class quiz

public Area asArea (Area s) {
    return s;
}
...
Rectangle r =
    new Rectangle (1,2,1,1);
Circle c = new Circle (1,1,3);
Area s1 = r;  // A
Area s2 = c;  // B
s2 = r;     // C

__D__ x = asArea (r);
__E__ y = asArea (s1);

s1 = c;  // F
s1 = s2; // G
r = c;   // H
r = s1;  // I

• What is the static type of s1 on line A?
• What is the dynamic class of s1 when execution reaches A?
• What is the static type of s2 on line B?
• What is the dynamic class of s2 when execution reaches B?
• What type should we declare for x (in blank D)?
• What is the dynamic class of x?
• What type should we declare for y (in blank E)?
• What is the dynamic class of y?
• Which of the assignments on lines F-I are well typed?