Index of /~cs61b/lectures/08

Name	Last modified	Size	Description
Parent Directory	24-Sep-98 21:22	-	

```
                    CS61b: Lecture 8
                   Friday, September 10
ANNOUNCMENTS:
-------------
Warning: The binary search procedure in the book (and lecture note 07) does
  not work for empty vectors.
The 61b waiting list is empty.  Remaining people were put in section 24.  If
  you want to switch sections, you my try to attend another section, but will
  have no guarantee of getting a seat.  Those on the roster have priority.

PASSING PARAMETERS (REVIEW)
---------------------------
Inside the Java implementation, there is a stack of "activation records."
Each time a method is invoked, Java adds an activation record to the stack.

As in Scheme, all parameters are passed by "value" meanthing that they
are copied.  However, if the parameter is an object (which we draw as a
pointer to that object), only the pointer is copied -- the object is shared.

 static void brokenInc (int x) {  |  int a = 1;          |    After inc calls:
    x = x + 1;  }                 |  brokenInc(a);       |
                                  |  Now a is:           |    a: 1
  class IntBox {                  |                      |
    public int value;             |                      |
    public IntBox(int i) {        |                      |
     value = i; }                 |                      |
  }                               |                      |
                                  |  IntBox b =          |
 static void workingInc           |    new IntBox(1);    |
    (IntBox iw) {                 |                      |
    iw.value = iw.value + 1;      |  WorkingInc(b);      |     ---------
  }                               |  Now b.value is:     |  b->|value:2|
                                  |                      |     ---------
                                  |   IntBox c =         |
  void broken2Inc (IntBox iw) {   |     new IntBox(1);   |     ---------
    iw = new IntBox(iw.value+1);  |   broken2Inc(c);     |  c->|value:1|
                                  |   Now c.value is:    |     ---------

A method can modify an object that one of its parameters points to.
A method can assign a parameter to a new value, but the outside world won't see it.

STATIC METHODS (REVIEW)
-----------------------
Most methods get one "special" argument, which is the "this" object.
E.g., when drawing a picture of the activation frame for deposit, include "this."

        Account acct = new Account (100);
```

```
        acct.deposit(50);
```

Some functions don't "belong" to an object; no "this" is needed. Use "static."

```
       int gcd(int x, int y) { ... }
```

A static method belongs to a class, not to an object:

	Static	Non-Static
invoke	ClassName.method(...)	ObjectName.method(...)
inside the class	method(...)	method(...) // same "this"

Common javac error: Can't call a non-static method from a static one.

Static recursive methods

```
  int fact (int x) { if (x == 0) return 1; else return (n * fact(n-1)); }
```

LINKED LISTS IN JAVA (A RECURSIVE DATA TYPE)

To build linked lists in Java, we separate two different problems:
building the nodes in the list, and building the entire list.

Each of the nodes is an object, to we use an "object factory," i.e., a class.

```
        -------------      As in Scheme, the first thing is some kind
        |  3  |  ---|-->    of element and the second is a pointer
        -------------
```

```
   class ListNode {                  // ListNode is a recursive type
     public int item;
     public ListNode next;           // Here we're using listNode before
   };                                // We've finished declaring it.

   ListNode l;  /* declaration of a "reference to" a ListNode */

   ListNode l1 = new listNode();     // What value does item have?
   ListNode l2 = new ListNode();     // What value does next have?
   ListNode l3 = new ListNode();
   l1.item = 1;   l2.item = 2;   l3.item = 3;

        -------------       -------------       -------------
    l1->|  1  | ? ->|   l2->|  2  | ? ->|   l3->|  3  |  ?->|
        -------------       -------------       -------------
```

To fill in the pointers between them and link them together:

```
   l1.next = l2;  l2.next = l3;
```

What about the last node? We need a "null" pointer for that.
There is a Java keyword to refer to the null reference for any type of object.

```
   l3.next = null;
```

BASIC OPERATIONS ON LISTNODES

To simply programming, we might add constructors:

```
  ListNode(int obj) { item = obj;   next = null;  }
  ListNode(in obj, ListNode n) {  item = obj;   next = n;  }
```

Java will no longer allow "new ListNode()", unlesss we define a 0-arg constructor.

```
We can now get the effect of "cons" using the constructors.

   ListNode l1 = new ListNode(1, new ListNode(2, new ListNode (3)));

We can modify lists as well:

   set-car!           l1.item = 0;
   set-cdr!           l1.next = new ListNode(4);

RECURSION ON LISTS
------------------
The nth operation in Scheme (called ptrTo here) can be written recursively:

  public ListNode ptrTo (int position) {
    if (position < 1) {  return null;  }
    else if (position == 1) { return this; }
    else if (next == null) { return null; }  // or error
    else { return next.ptrTo(position-1); }
  }
```