Index of /~cs61b/lectures/10

```
   Name                    Last modified     Size  Description
```

```
   Parent Directory        24-Sep-98 21:22      -
   Lecture10.java          16-Sep-98 00:13     2k
```

```
                    CS61b: Lecture 10
                   Wednesday, September 16
ANNOUNCMENTS:
-------------
Any partnerless people meet near the stage after lecture.
Discussion sections: 124 and 122 are being merged, Thu @ 2pm in 320 Soda.

VECTORS REVISITED
-----------------
Chapter 10 of AW covers vectors in more detail. An example is transpose of
element at k and k-1 (similar to swap).  Here is their first implementation:

private void swap (Vector v, int k) {
   if (k != 0) {
     v.insertElementAt(v.elementAt(k), k-1);
     v.removeElementAt(k+1);
   }
}

This is very slow.  Why?  AW improves on it later, using code more like our swap.
Conclusion: Vectors insert and remove operations are nice, but use with care.

ARRAYS (LESS POWERFUL, OFTEN THE BASIS FOR VECTOR IMPLEMENTATIONS)
------
An array is a another basic data stucture in Java that can contain other data.
 * Similar to Vectors, but are not a class in the library; they are built-in.
 * Unlike Vectors, but do not grow/shrink, and cannot insert/remove from middle.

int [] a = new int[3];          // array with 3 elements, indexes 0, 1, 2
a[0] = 1; a[1] = 2; a[2] = 3;   // set values in a
int y = a[0];                   // get the 0th value of a
int [] b;    // says b is an array, but it doesn't exist
b[0] = 1;    // error
b = new int[3];  // now b exists, but it's elements only have default values

   a:-->   ----------------          b: -?->
           |    |    |    |
           ----------------

PASSING ARRAY TO AND FROM METHODS
---------------------------------
Arrays are objects; they are passed by copying the reference (sharing the array)
 public static int [] copy (int [] v) {
     int [] result = new int [v.length];
     for (int i = 0; i < v.length; i++) {  result[i] = v[i];  }
     return result;
```

```
  }

  int [] c = copy(a);
```

INITIALIZATION

Shorthand for creating an array and initializing all of its elements.

```
    int [] c = {1, 2, 3};
    c = {2, 4, 6};        // error: {...} only allowed in initial assignment
```

OBJECTS

You may create an array of any type:

```
    Account [] d = new Account [2]; // array of references to Accounts
    Object [] e = new Object [2]; // array of references to any Object
```

d is homogenous (only contains Account ref's); e is heterogenous, like Vectors. All the references are null in both arrays. To create on with actual objects:

```
    Object [] f = {new Account(100), new Integer(3)};
```

MULTIDIMENSIONAL ARRAYS

A multidimensional array is an array of arrays--it need not be rectangular.

```
   int [][] x = new int [3][4];       // x is a rectangular array of arrays
   int [][] y = new int [1][];
   y[0] = new int [2];
   y[1] = new int [1];                // y is non-rectangular
   y[1][0] = y[0][1];
   int [] z = copy(y[0]);       // y[0] is a 1-dimensional array of 2-elements
```

Initializers can be used on multidimensional arrays by nesting them.

```
   int [][] z = {{1, 3}, {2, 4, 6}};
```

It is often an important invariant if and array is rectangular:

```
     for all i,j  0 <= i,j < x.length, x[i].length == x[j].length
```

GROWING ARRAYS

Arrays cannot change size, but we can copy them to larger ones.

```
  public static int [] resize (int [] v, int newSize, int init) {
      int [] result = new int [newSize];
      int k;
      for (k = 0; k < v.length; k++) { result[k] = v[k]; }
      for (; k < newSize; k++) { result[k] = init; }
      return result;
   }

    d = resize(d, d.length*2+1, 9);
```

EQUALITY OPERATIONS (AN ASIDE)

In Scheme you learned about eq? and equal? AW talks about "equals" in Chapter 8.

```
  x == y       always returns true iff y and y are the same object, like eq?
```

```
      o1 ----> ----------           o3---->----------     o1 == o2
      o2 ----> | bal:50 |                  | bal:50 |     o1 != o3
               ----------                  ----------
```

There is another notion of equality that you've seen called "equals."

  o1.equals(o2) by default, returns true iff o1 and o2 are the same object

But a classes can define equals (override default) to mean different things.
Strings, for example, define it to mean "the strings have the same characters."

Common definitions of equals:
1) the default is called pointer equality (same as ==)
2) another common case is structural equality
   i.e., two objects are "equals" if all their fields are "equals"
3) logical equality

By "logical equality" we mean that you can't tell the difference between the two objects with any other methods, e.g.,
* two "Fractions" may be considered equal if they represent the same rational number, even if they are not in reduced form.