On (computational) simplicity

We are trying to teach not just Java, but how to think about problem solving.

Computer science has its field called computational complexity; mine is called computational simplicity. – Prof. Gries

Most of us don’t write perfect essays in one pass, and coding is the same: writing requires revising; programming requires revising. If your code is getting convoluted and you have trouble understanding it: stop and look for a better way. If your code is getting longer and longer, with no end in sight: stop and look for a better way.

Learn to keep things simple, to solve problems in simple ways. This sometimes requires a different way of thinking.

A key point is to break a problem up into several pieces and do each piece in isolation, without thinking about the rest of them. Our methodology for developing a loop does just that.

Array: object that stores lists of things.

Holds a fixed number of values of a declared type. (So an will always hold 4 int values.)

The type of array a0 is int[]

Store its name in a variable (as always).

Basic form of a declaration:

\[
<\text{type}> <\text{variable-name}> ;
\]

Does not create array, it only declares x. x’s initial value is null.

Elements of array are numbered: \(0, 1, 2, \ldots, \text{x.length} - 1\)

Notes on array length

We write x.length, not \(x.length()\), because length is a field, not a method.

Length field is final: an array’s length (field or actual number of items) cannot be changed once the array is created.

We omit this field in the rest of the pictures.

The length is not part of the array type, which is int[].

This means that an array variable can be assigned arrays of different lengths; \(x\) could later hold the name of a seven-item int array. (But not the name of a seven-item double array.)

Array initializers

Instead of

\[
\text{int[]} c = \text{new int}[5];
\]


Use an array initializer:

\[
\text{int[]} c = \text{new int}[\{4, 4, 7, 6, 9\}];
\]

array initializer: gives initial values for the array items. Values must have the same type, in this case, int. Length of the array is the number of values in the list; so …

... must omit expression between brackets. Sometimes, can even omit the “\(\text{new <type>}[]\)” part (see pg 274).
**Use of an array initializer**

```java
public class D {
    public static final String[] months = new String[] {
}
```

**Differences between array and Vector ("classier", fbofw)**

<table>
<thead>
<tr>
<th>Declaration:</th>
<th>int[] a;</th>
<th>Vector v;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements of a</td>
<td>int values</td>
<td>Elements of v: any Objects</td>
</tr>
<tr>
<td>Creation:</td>
<td>= new int[n];</td>
<td>= new Vector();</td>
</tr>
<tr>
<td>Array always has n elements</td>
<td>Number of elements can change</td>
<td></td>
</tr>
<tr>
<td>Reference element:</td>
<td>a[e]</td>
<td>v.get(e)</td>
</tr>
<tr>
<td>Change element:</td>
<td>a[e]= e1;</td>
<td>v.set(e, e1);</td>
</tr>
</tbody>
</table>

**Procedure swap for swapping array elements**

```java
public class D {
    /** = Swap b[h] and b[k] */
    public static int swap(int b[], int h, int k) {
        int temp = b[h];
        b[h] = b[k];
        b[k] = temp;
        return temp;
    }
}
```

**Linear search**

```
**Linear search**
```

```java
public class D {
    // * = index of first occurrence of c in b[]
    public static int findFirst(int c, int[] b) {
        int i = 0;
        // invariant: c is not in b[0..i-1]
        while (b[i] != c) {
            i += 1;
            // b[i] = c and c is not in b[i..i-1]
        } return i;
    }
}
```

**Non-uniform randomness from uniform randomness: It's a kind of linear search!**

```java
public class D {
    /** = a random int in 0..p.length-1, assuming p.length > 0. */
    /** The (non-zero) prob of int i is given by p[i]. */
    public static int roll(double[] p) {
        double r = Math.random(); // r in [0,1)
        // inv: r not in segments looked at (segments 0..i-1)
        // and iEnd is the end of (just after) segment i
        while (r >= p[iEnd]) {
            iEnd = iEnd + p[iEnd];
            i += 1;
            // r is in segment i
        }
        return i;
    }
}
```