The median again

The steps of our algorithm:

- Read the size of the list, N.
- Declare and instantiate an array of integers, "list".
- Read the elements of list.
- Check each element to see if it's the median.

The last step goes like this:

```
for (int i = 0; i < N; i++)
  if (/* list[i] is the median */) {
    System.out.println("median = " + list[i];
    break;
  }
```

Exercise

Write a method to count the members of an array that are smaller than some value:

```
int countSmaller (int refVal, int[] list)
```

An answer for the exercise

```
// countSmaller: return the number of entries
// in list that are smaller than refVal.
int countSmaller (int refVal, int[] list) {
  int count = 0;
  for (int i = 0; i < list.length; i++)
    if (list[i] < refVal)
      count++;
  return count;
}

// or ...
    count += (list[i] < refVal) ? 1 : 0;
```

Finishing off the median

Now that we have countSmaller( ),

```
if (/* list[i] is the median */)
```

becomes

```
if (countSmaller(list[i], list) == list.length/2)
```

-- so our median-finding loop is:

```
for (int i = 0; i < list.length; i++)
  if (countSmaller(list[i], list) == list.length/2)
    System.out.println("median = " + list[i];
    break;
  }
```

Another exercise

Rewrite all this median-finding stuff as a method:

```
int median (int[] list)
```

- Does it return the value of the median or the index of the list element that is the median?

The parameter of main( )

```
public static void main (String[] args)
```

If your program contains:

```
for (int i = 0; i < args.length; i++)
  System.out.println(args[i]);
```

then it will print the arguments of the command that started the program.
How does the command ***get*** arguments?

```
java Program hi there
```

- or in CodeWarrior, set "Parameters" in the "Java Target" area of the Project Manager window.

Two-dimensional arrays

A table of marks:

	A1	A2	Test
Jim	2	8	15
Mary	9	7	18

Jim's row is row 0; Mary's row is row 1.
A1's column is column 0; A2's column is column 1.

	col 0	col 1	col 2
row 0	(0,0)	(0,1)	(0,2)
row 1	(1,0)	(1,1)	(1,2)

We can't do this with a simple array (or Vector); we need something 2-D.

Using a two-dimensional array

Declaring:

```
int[][] table;
```

Creating:

```
table = new int[2][3]; // [row size][column size]
```

Filling with values:

```
for (int row = 0; row < table.length; row++)
  for (int col = 0; col < table[0].length; col++)
    table[row][col] =
        Integer.parseInt(in.readLine());
```

Arrays *vs* Vectors

1. Vectors can grow (and shrink). Arrays cannot.

2. Vectors are just another class. Arrays are a fundamental part of the Java language.

3. Vectors can contain a mixture of types. An array can contain only one type of element.

4. Vector elements must be (references to) objects. An array's element type can be either a class type or a primitive type.

- Vectors can be inefficient, while sometimes arrays are not inefficient in the same ways. But "inefficiency" is always "inefficient in the context of a particular application", and anyway we don't have the tools even to talk about efficiency properly yet.

An array example: the transpose

A 2-D array corresponds to a mathematical ***matrix***:

11	12	13
21	22	23
31	32	33

One simple matrix operation ***transposes*** the matrix elements:

11	21	31
12	22	32
13	23	33