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

Our first solution

```
for (int i = 0; i < A.length; i++)
    for (int j = 0; j < A[0].length; j++) {
        // Swap A[i][j] and A[j][i].
        int temp = A[i][j];
        A[i][j] = A[j][i];
        A[j][i] = temp;
    }
```

What's wrong?

Fixed

```
for (int i = 0; i < A.length; i++)
    for (int j = 0; j < i; j++) {
        // Swap A[i][j] and A[j][i].
        int temp = A[i][j];
        A[i][j] = A[j][i];
        A[j][i] = temp;
    }
```

Does it work if you write " `j <= i`"?

Testing

Remember the family-name problem. Suppose we have a solution:

```
String parseFamilyName (String name)
```

Imagine we've tested it, with these results:

name	return value	OK?
Clarke Jim	**Clarke**	**yes**
Chan Mary	**Chan**	**yes**
MacDonald John	**MacDonald**	**yes**
O'Shaughnessy Pat	**O'Shaughnessy**	**yes**
DuMaurier Diane	**DuMaurier**	**yes**
Thant U	**Thant**	**yes**
Hammarskjold Dag	**Hammarskjold**	**yes**
Tryggve Lie	**Tryggve**	**yes**

Is it OK?

Is our testing convincing?

- a very short name: length 1, 2
- given name missing
- given name missing but still a pair of blanks after the family name
- given name missing but a single blank after the family name
- a blank within the family name
- a blank after the given name
- a pair of given names
- a given name containing a double blank

Bad data

Should we test these cases?

- a blank at the beginning
- two blanks at the beginning
- the empty string
- a single blank

They're not covered by the specifications – but should they be?

Testing is hard

It's boring, too – but you have to think about it from the start.

Testing rules

- Design test cases ***first***, before coding.
- Use testing tools to help prepare test cases.
- Don't test your own work.

What if you're the only programmer?

Test ***all*** the cases?

Testing integer multiplication:

- 2^{32} integers
- $2^{32} \times 2^{32} = 2^{64} \cong 10^{21}$ different products
- at 1 nanosecond per test, $10^{21-9} = 10^{12}$ seconds, or about 100,000 years

And first, someone made a table of the right answers.

So, how to test?

You can't test all cases, but you can try for two kinds of completeness:

<u>Complete coverage of the kinds of data</u>

- positive, 0, negative
  - other boundary values: blank character, empty string, empty list
- small (0, 1, 2) and large
  - or short and long
- extremes: selected value found at beginning or end of list
- duplicates
  - median of list with pairs, triples, or ***all*** the same
- special sets
  - sorted, reverse-sorted, all the same

Another kind of completeness

<u>Complete coverage of your code</u>

- Your test cases should cause every line of your program to be executed.
  - Test tools can help here.
- For example: both branches of an "if" statement.
- Each loop should be executed each of 0 times, 1 time, and >1 times.

And if it's not possible to exercise a line, what's it doing in your program?

The two kinds of completeness interact.
While checking code coverage, you'll think of things to add to the data coverage.

More testing homilies

Justify your test cases.

- If there isn't a reason for a test case, discard it.

Test early.

- Design your test cases before you write your code.
- (But look at them again after you code.)

Built-in testing

Make it easier to test your code.

- Write toString( ) methods.
- Include print statements in your code.
  - Guard them with "if (testing)" so you can control the output.
- Write a main( ) method for each class.

Don't trust yourself

Get an enemy to test your code.

(Why? If you don't understand the problem when you write your code, you can't test it either.)