Extreme OOP

A funny and useful exercise for smart programmers of all ages

The object of the game is to develop a program, following two sets of rules.

TDD Rules

1. You can't write production code without a failing test
2. You can't have more than one failing test at a time
3. You can't write a new test until your code is properly refactored

Extreme OOP Rules

1. Use only one level of indentation per method.
2. Don't use the `else` keyword.
3. Wrap all primitives and strings.
4. Use only one dot per line.
5. Don't abbreviate.
6. Keep all entities small.
7. Don't use any classes with more than two instance variables.
8. Use first-class collections.
9. Don't use any getters/setters/properties

Explanation of the rules

Use only one level of indentation per method.

Like this:

```
void collectRows(StringBuffer buffer) {
  for(int i = 0; i < 10; i++)
    doSomething(buf, i);
}
```

Don't use the `else` keyword.

Ever. You may use an early return. Other techniques like polymorphism and null objects can help.

Wrap all primitives and strings

No method should have an argument that is either a primitive type or a String.

Use only one dot per line

The Pryce-McKinnon example: don't do like this:

```
dog.getBody().getTail().wag();
```

Do like this:

```
dog.expressHappiness();
```

and let the implementation decide what this means.

Don't abbreviate

Try to keep method and class names to one or two words.

Keep all entities small.

No file longer than 50 lines, no package with more than 10 files.

Don't use any classes with more than two instance variables.

Instead of doing this,

```
class Name {
  String first;
  String middle;
  String last;
}
```

You may do this:

```
class Name {
  Surname family;
  GivenNames given;
}
class Surname {
  String family;
}
class GivenNames {
  List names;
}
```

Use first-class collections

If a class contains a collection, then it must be the only instance variable in that class.

Don't use any getters/setters/properties

Tell, don't ask. Don't meddle in the internals of other objects. If the other object has the data, then let the other object perform the operation.

How to develop extremely OO code with TDD

See the *Mock Roles, not Objects* paper.

The exercise

We are to develop an interpreter for (something similar to) Commodore 64 BASIC

Story: An empty program produces no output. Acceptance:

```
input:
(empty)
output:
(empty)
```

Story: A bare "print" statement produces a single newline. Acceptance:

```
input:
PRINT
output:
(a single newline)
```

Story: A "print" statement can have a constant string as an argument. The output is the constant string. Acceptance:

```
input:
PRINT "Hello, World!"
output:
Hello, World!
```

Story: Two or more statements in a sequence are executed one after the other

```
PRINT "Hi"
PRINT "There"
PRINT "!"
output:
Hi
There
!
```

Story: The “print” statement can output number constants.

```
PRINT 123
output:
123
PRINT -3
output:
-3
```

Story: A single letter is a variable. The print statement can print its value. The default value for a variable 0

```
PRINT A
output:
0
```

Story: An assignment statement binds a value to a variable.

```
input:
A=12
PRINT A
output:
12
```

Story: Two numeric constants can be added together.

```
PRINT 3 + 7
output:
10
```

Story: A numeric expression can have more than two terms.

```
PRINT 4 + 4 + 12
output
20
```

Story: A numeric expression can be built with variables and/or constants

```
A=2
B=7
PRINT A + 1
PRINT A + B
PRINT 99 + B
output:
3
9
106
```

Story: Two numeric expressions can be subtracted

```
PRINT 1 - 2
output:
-1
```

Story: Expressions can be parenthesized

```
PRINT 1 - (2 + 3)
output:
```

-4

References

The idea for the whole exercise is due to:

- Jeff Bay, Object Calisthenics, in *The ThoughtWorks Anthology*, Pragmatic Bookshelf, 2008.

The C64 Basic kata was mentioned in the Agile Finland wiki:

- `http://wiki.agilefinland.com/?CodingDojo20070606`

The rationale for the “don’t use any getters” rule is well explained in

- Andy Hunt, Dave Thomas: *Keep It DRY, Shy, and Tell the Other Guy*
  `http://media.pragprog.com/articles/may_04_oo1.pdf`

The mock-objects approach to TDD is explained best in:

- *Mock Roles, not Objects* by Steve Freeman, Nat Pryce, Tim Mackinnon, Joe Walnes.
  `http://www.jmock.org/oopsla2004.pdf`