CLOS, the Common Lisp Object System

Christopher Mark Gore

http://www.cgore.com

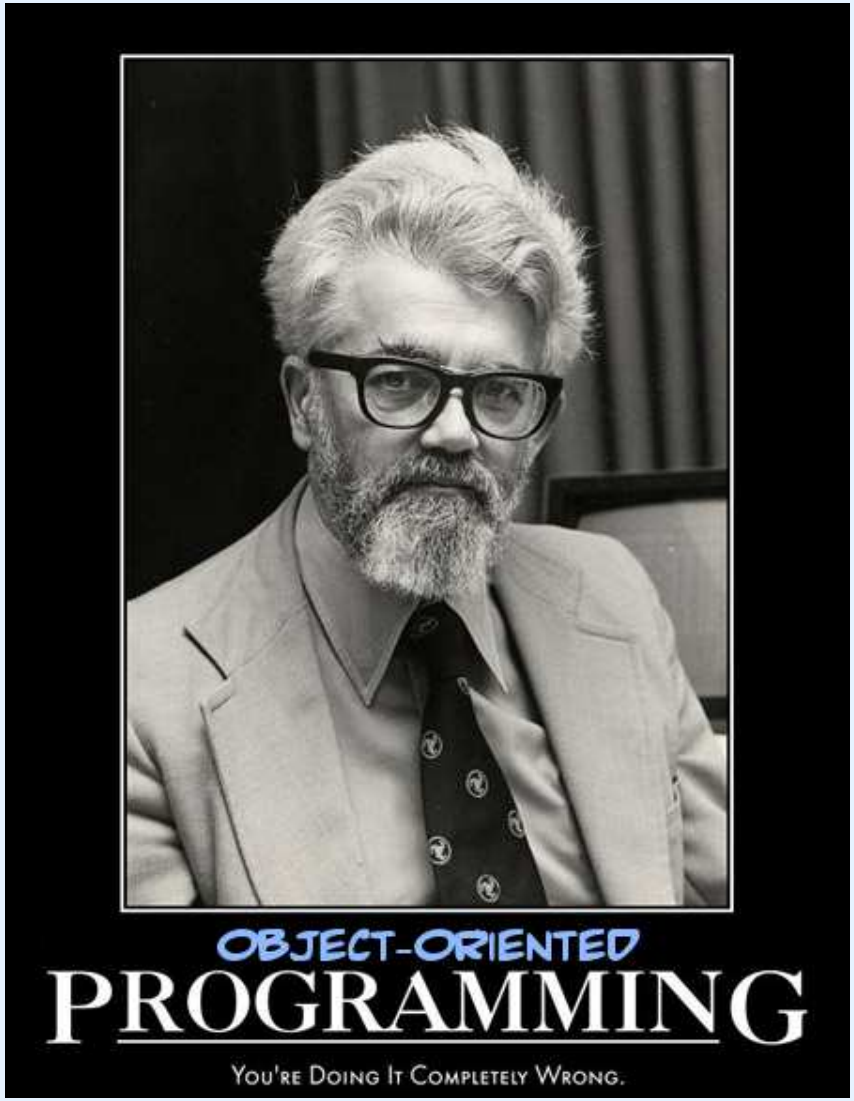
cgore@cgore.com

Thursday, February 6, AD 2014

Getting Started with Common Lisp

1. Install Linux.
   `http://aptosid.com`
2. Install SBCL and some libraries.
   ```
   apt-get install sbcl{,-doc,-source} \
   cl-{asdf,cffi}
   ```
3. Install Emacs and SLIME *(Not strictly required.)*
   ```
   apt-get install emacs{,-goodies-el} cl-swank \
   cl-swank slime common-lisp-controller
   ```

Lisp + OOP > OOP − Lisp



OOP isn't *The* Answer

OOP is a useful tool, but it isn't the final solution to all things programming. It won't solve world hunger, but it does solve a restricted subset of the problem. Other things that are sometimes useful tools, but aren't **the** answer:

1. FP,
2. Lack of side effects,
3. Unit tests/TDD/BDD,
4. Type systems,
5. Monads,
6. Your favorite thing in programming,
7. ~~Lisp.~~ *[Yes it is.]*

Nouns and Verbs

- Nouns are how our brain works with things.
  *The **cat** was asleep in the **hallway**.*
  `cat[27].goToSleepInLocation(hallway[3]);`
- Verbs are how our brain works with actions.
  *He **murdered** her in cold blood!*
  `(with-person (the-man) (murder the-woman))`

Other Fun Parts of Speech . . .

- Adjectives describe nouns.
  *The **big**, **old**, **yellow** house burned to the ground.*
  `<house size="big" age="old" color="yellow"/>`
- Pronouns *(anaphors)* are shortcuts for nouns.
  ***We** walked down the street to meet **him**.*
  `(a?if him (person-to-meet?)  (go-to-meet him))`

. . . Other Fun Parts of Speech

- Adverbs change verbs.
  *He **quickly** ran down the street.*
- Prepositions links nouns and pronouns to other words.
  *The book is **beneath** the table.*
- Conjuctions link words, phrases, and clauses.
  *I ate the pizza **and** the pasta.*
  `int i = 12; i++;`
- Interjections convey emotion.
  ***Hey!** Put that down!*

Lisp + OOP > Lisp?

[Opinionated opinion:] Java or C++ style OOP doesn't help if you already have Lisp, and probably hurts, but the CLOS does help if you have the right sort of problem.

- Lisp is good at modeling computation.
- Functional programming is good at modeling verbs,
- Object-oriented programming is good at modeling nouns.
- CLOS allows FP for verbs and OOP for nouns to interact easily, with neither being the ***King Of All The Words***. (Cf. Steve Yegge's essay, *Execution in the Kingdom of Nouns*, `http://steve-yegge.blogspot.com /2006/03/execution-in-kingdom-of-nouns.html`).

OOP: How Classes See Their Methods

FP: How Functions See Their Data

The Basic Components in CLOS

- *Classes* model nouns.
- *Instances* are specific occurrences of nouns.
- *Generics* model verbs.
- *Methods* implement generics for specific classes.

The first two, classes, and instances, work as expected from any other normal OOP language. Generics and methods work quite differently though.

DEFCLASS

We define new classes with the `defclass` macro.

`(defclass` *`class-name`* `(`*`superclass-names`*`) (`*`slots`*`))`

Some examples:

```
(defclass point () (x y))
(defclass shape () ()) ; An abstract base class.
(defclass rectangle (shape) (p q))
(defclass circle (shape) (center radius))
```

We typically want to provide more for the slot definitions.

```
(defclass better-point ()
          ((x :accessor x :initarg :x
              :initform 0.0 :type float)
           (y :accessor y :initarg :y
              :initform 0.0 :type float)))
```

MAKE-INSTANCE

We create new instances of a class with `make-instance`. *(The following examples assume more thorough slot definitions.)*

```
(let* ((origin (make-instance 'point))
       (p1 (make-instance 'point :x 1.0 :y 12.5))
       (p2 (make-instance 'point :x 5.0 :y 10.0))
       (r1 (make-instance 'rectangle :p p1 :q p2))
       (c1 (make-instance 'circle :center origin
                                  :radius 12.5)))
  (x p1) ; Returns 1.0
  ; Do more stuff ... )
```

DEFGENERIC

We define new generics with `defgeneric`.

`(defgeneric` *`generic-name lambda-list`*`)`

Some examples:

```
(defgeneric min-x (thing))
(defgeneric max-x (thing))
(defgeneric min-y (thing))
(defgeneric max-y (thing))
(defgeneric height (thing))
(defgeneric width (thing))
(defgeneric area (thing))
```

These define the general layout of a set of methods all with the same name, basically something similar to their "function signature", but don't really do much else. *(SBCL will implicitly create them for you, with a warning.)*

DEFMETHOD

We define new methods with `defmethod`.

```
(defmethod min-x ((r rectange))
  (min (x (p r)) (x (q r))))
```

Implement `max-x`, `min-y`, and `max-y` in a similar manner.

```
(defmethod height ((r rectangle))
  (- (max-y r) (min-y r))
(defmethod width ((r rectangle))
  (- (max-x r) (min-x r)))
(defmethod area ((c circle))
  (* pi (expt (radius c) 2)))
(defmethod area ((r rectangle))
  (* (height r) (width r)))
```

Questions?