

Object-Oriented Programming using the Common Lisp Object System

Daniel A. Matysiak

What is CLOS?

- CLOS (Common Lisp Object System) is the object-oriented programming standard for Common Lisp. It facilitates programming using a sophisticated object-oriented model.

Functionality

- generic functions
- multiple inheritance
- declarative method combination
- meta-object protocol

Defining a Simple Class

```
(defclass <name> (<list-of-superclasses>)  
  ((<slot>)  
   (<slot>)))
```

- Example:

```
(defclass cat ()  
  ((n-legs)  
   (name)))
```

Basic Instantiation

```
(make-instance 'cat)
#<CAT {400360B5}> ; our cat object

; both slots are unbound, or free
(setf foofoo (make-instance 'cat))

(setf (slot-value foofoo 'name) "foofoo")

; 'name' is bound
(slot-value foofoo 'name)
"foofoo"
```

Inheritance

```
(defclass animal ()
  ((alive?)))

(defclass mammal ()
  ((environment)))

(defclass cat (animal mammal)
  ((n-legs)
   (name)))

(setf foofoo (make-instance 'cat))
(setf (slot-value foofoo 'alive?) t)

; return what's in slot 'alive?'
(slot-value foofoo 'alive?)
T
```

Slot Options

```
(defclass cat ()
  ((name :initarg      :give-name
         :type         string
         :reader       get-name)
   (age  :initform    0)
   (owner :accessor   owner
         :allocation  :class
         :documentation "The owner of all cats")
   (state :writer     secret-state)))
```

Using Slots

```
(setf foofoo (make-instance 'cat :give-name "foofoo"))  
(get-name foofoo)  
"foofoo"
```

```
(setf (owner foofoo) 'sergei)  
(owner foofoo)  
SERGEI
```

```
(setf fifi (make-instance 'cat :give-name "fifi"))  
(owner fifi)  
SERGEI
```

```
(secret-state 'dead foofoo)
```

Beyond Accessors and Mutators

- methods can be defined to respond to messages in more complex ways
- extends to allow polymorphic response dependent on message passed
- unlike simpler and less capable object systems, CLOS does not associate methods with a particular class

Defining Methods

- defined in much the same way as using defun, but with the option of type-specified parameters (specialized lambda list)

```
(defmethod (param1 (param2 Type))  
  ...)
```

Single Dispatch Methods

- if you're an optimist:

```
(defmethod get-age ((obj cat))  
  (slot-value obj 'age))
```

- if you're a pessimist:

```
(defmethod years-left ((obj cat))  
  (- 15 (slot-value obj 'age)))
```

Multiple Dispatch Methods

- classes

```
(defclass Circle (2d-coord)
  ((radius :accessor my-radius)))
```

```
(defclass Rectangle (2d-coord)
  ((length :accessor my-length)
   (width :accessor my-width)))
```

- methods

```
(defmethod area ((obj Circle))
  (* pi (expt (my-radius obj) 2)))
```

```
(defmethod area ((obj Rectangle))
  (* (my-length obj) (my-width obj)))
```

- calling (*area object*) generic function will select the closest matching method based on parameters

Further Down the Road: More Advanced Features of CLOS

Specifying Generic Functions

- if you choose to explicitly define a generic function, c'est possible avec *defgeneric*

```
(defgeneric Name (param1 param2)
  (:documentation "A sample generic function")
  (:method ((obj1 Type1) (obj2 Type2)) (function body))
  (:method-combination progn)
  (:argument-precedence-order param2 param1))
```

Method Combination

- primary method

```
(defmethod method-name ((n number))  
  n)
```

- before methods

```
(defmethod method-name :before ((n integer))  
  'integer)
```

```
(defmethod method-name :before ((n rational))  
  'rational)
```

- after methods

```
(defmethod method-name :after ((n integer))  
  'integer)
```

```
(defmethod method-name :after ((n rational))  
  'rational)
```

Method Combinations (cont'd)

- around methods

```
(defmethod method-name :around ((n integer))  
  (call-next-method))
```

```
(defmethod method-name :around ((n complex))  
  'sorry)
```

A Taste of the Metaobject Protocol

- MOP makes it possible to peek into the internals of CLOS, and if you so please, modify the behavior of CLOS
- not standardized as a part of the CLOS specification, but implemented in virtually every major CL interpreter
- Examples of MOP functions:

CLASS-DIRECT-SUBCLASSES
CLASS-DIRECT-SUPERCLASSES
CLASS-DIRECT-SLOTS
CLASS-DIRECT-METHODS
CLASS-PRECEDENCE-LIST

Wrapping Things Up

- CLOS doesn't suck
- More capable than other well known systems
- Complex, yet powerful
- Go try it out!