Object Oriented Programming

- What is it?
- What does it have to do with Scheme?

Object Oriented Programming

- Is *not* the same thing as C++
- Object oriented programming = objects + inheritance
- Objects

  Encapsulated data

  A language with objects is called *object based*

Object Oriented Programming

- Inheritance

  Mechanism by which one type of object can use the attributes of another type

  A language with objects *and* inheritance is called *object oriented*

Object Oriented Scheme

- Objects?
- Ineritance?

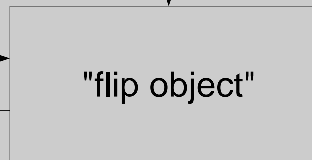
Basic Object System

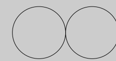
- Recall `make-flipper`
- `make-flipper` created an object (in this case, a procedure)
- That object received messages and returned other procedures, depending on the message
- Those other procedures (“methods”) are applied (perhaps to arguments) to do things to the object (i.e., change its state)

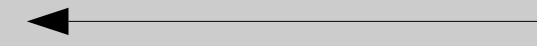
Basic Object System

This procedure is the one that lets us actually change the state of the object (this procedure is sometimes called a "method").

message in







procedure out

Example

- The following procedure creates a simple object called a “speaker”

```
(define (make-speaker)
  (lambda (message)
    (cond ((eq? message 'say)
           (lambda (self stuff)
                   (print stuff)))
          (else (no-method ``speaker''))))))
```

Example

- We get methods from objects by asking for them

```
(define (get-method object message)
  (object message))
```

Example

- If an object does not recognize the message, it uses `no-method`, which returns something that our object system will recognize

```
(define (no-method name)
  (list 'no-method name))

(define (no-method? x)
  (if (pair? x)
      (eq? (car x) 'no-method)
      false))

(define (method? x)
  (not (no-method? x)))
```

Example

- `ask` gets the method and applies it to arguments

```
(define (ask object message . args)
  (let ((method (get-method object message)))
    (if (method? method)
        (apply method (cons object args))
        (error ``No method'' message (cadr method)))))
```

Example

```
1 ]=> (define chris (make-speaker))
CHRIS

1 ]=> (ask chris 'say
            '(its time to snowboard))
(ITS TIME TO SNOWBOARD)
```

Inheritance

- Define an object type to be a kind of some other object
- A "lecturer" is a kind of speaker with a `lecture` method

```
(define (make-lecturer)
  (let ((speaker (make-speaker)))
    (lambda (message)
      (cond ((eq? message 'lecture)
             (lambda (self stuff)
               (ask self 'say stuff)
               (ask self 'say
                    '(you should not be taking notes))))
            (else (get-method speaker message))))))
```

Inheritance

```
1 ]=> (define sharon (make-lecturer))

1 ]=> (ask sharon 'say
           '(its time to rock and roll))

1 ]=> (ask sharon 'lecture
           '(its time to rock and roll))
```

Inheritance

```
1 ]=> (ask sharon 'say '(its time to rock and
(its time to rock and roll)

1 ]=> (ask sharon 'lecture '(its time to rock
(its time to rock and roll)
(you should not be taking notes)
```

Inheritance

- To make a lecturer a kind of speaker, we gave it an internal speaker of its own
- If the message that the lecturer gets is not recognized, it passes it on to the internal speaker
- A lecturer can do anything a speaker can, but it can also do more
- The lecturer *inherits* the say method from speaker
- The speaker is a base class for the lecturer

Inheritance

- What is `self` used for?

Example

```
(define (make-arrogant-lecturer)
  (let ((lecturer (make-lecturer)))
    (lambda (message)
      (cond ((eq? message 'say)
             (lambda (self stuff)
               (ask lecturer 'say
                 (append '(it is obvious that) stuff))))
            (else (get-method lecturer message))))))

(define lumsdaine (make-arrogant-lecturer))
```

Example

```
1 ]=> (ask lumsdaine 'say
              '(its time to rock and roll))

1 ]=> (ask lumsdaine 'lecture
              '(its time to rock and roll))
```

Example

```
1 ]=> (ask lumsdaine 'say
            '(its time to rock and roll))
(it is obvious that its time to rock and roll)

1 ]=> (ask lumsdaine 'lecture
            '(its time to rock and roll))
(it is obvious that its time to rock and roll)
(it is obvious that you should not be taking r
```

Exercises

- Define an adolescent object that is a speaker but which prepends “like,” and appends “you know?” to every statement it says
- Define an adolescent-lecturer

Exercise

- Create an environment diagram for

```
(define lumsdaine (make-arrogant-lecturer))

(ask lumsdaine 'say '(scheme is cool))

(ask lumsdaine 'lecture '(scheme rules))
```