TODAY

Object Oriented Programming (OOP)

• Data Abstraction using procedures with state
• Message-Passing
• Object Oriented Modelling
  • Class diagrams
  • Instance diagrams
• Example: space wars simulation
OBJECT ORIENTED PROGRAMMING (OOP)

Terminology

• **Class**
  • specifies the common behavior of entities
  • in scheme, a "maker" procedure
e.g. cons in our previous examples

• **Instance**
  • A particular object or entity of a given class
  • in scheme, an instance is a message-handling procedure
    made by the maker procedure
e.g. foo or bar in our previous examples
OBJECT ORIENTED PROGRAMMING (OOP)

Terminology

Class Diagram

- class
- private state
- public messages

<table>
<thead>
<tr>
<th>PAIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>x:</td>
</tr>
<tr>
<td>y:</td>
</tr>
<tr>
<td>CAR</td>
</tr>
<tr>
<td>CDR</td>
</tr>
<tr>
<td>PAIR?</td>
</tr>
<tr>
<td>SET-CAR!</td>
</tr>
<tr>
<td>SET-CDR!</td>
</tr>
</tbody>
</table>

(\(def\)ine (my-cons x y))
(\(\lambda\) (msg) ...))

Instance Diagram

foo

PAIR
x: 3
y: bar

instance of a pair

bar

PAIR
x: 1
y: 2

instance of a pair

(define foo
(my-cons 3 (my-cons 1 2)))
DESIGN A SYSTEM USING OOP

Star Wars Simulator

• start by thinking about what kinds of objects we want (what classes, their state information, and their interfaces)
  • ships
  • planets
  • other objects

• extend it to particular instances of objects
  • Millenium Falcon
  • Enterprise
  • Earth
(define (make-ship position velocity num-torps)
  (define (move)
    (set! position (add-vect position velocity))
    'done)
  (define (fire-torp)
    (cond ((> num-torps 0) ...)
      (else 'out-of-torpedoes!)))
  (lambda (msg)
    (cond ((eq? msg 'position) position)
      ((eq? msg 'velocity) velocity)
      ((eq? msg 'move) (move))
      ((eq? msg 'attack) (fire-torp))
      (else (error "ship can't" msg))))
STAR WARS SIMULATOR
A Space-Ship Object

(define (make-ship position velocity num-torps)
  (define (move)
    (set! position (add-vect position velocity))
    'done)
  (define (fire-torp)
    (cond ((> num-torps 0) ...)
      (else 'out-of-torpedoes!)))
  (lambda (msg)
    (cond ((eq? msg 'position) position)
      ((eq? msg 'velocity) velocity)
      ((eq? msg 'move) (move))
      ((eq? msg 'attack) (fire-torp))
      (else (error "ship can't" msg))))
(define (make-ship position velocity num-torps)
  (define (move)
    (set! position (add-vect position velocity))
    'done)
  (define (fire-torp)
    (cond ((> num-torps 0) ...)
          (else 'out-of-torpedoes!)))
  (lambda (msg)
    (cond ((eq? msg 'position) position)
          ((eq? msg 'velocity) velocity)
          ((eq? msg 'move) (move))
          ((eq? msg 'attack) (fire-torp))
          (else (error "ship can't" msg))))))
STAR WARS SIMULATOR
Example Instances of Space-Ship Class

(define enterprise
  (make-ship (make-vect 10 10) (make-vect 5 0) 3))

(define war-bird
  (make-ship (make-vect -10 10) (make-vect 10 0) 10))

<table>
<thead>
<tr>
<th>SHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>position:</td>
</tr>
<tr>
<td>velocity:</td>
</tr>
<tr>
<td>num-torps:</td>
</tr>
</tbody>
</table>

position
velocity
move
attack

enterprise

<table>
<thead>
<tr>
<th>SHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos: (vec 10 10)</td>
</tr>
<tr>
<td>vel: (vec 5 0)</td>
</tr>
<tr>
<td>num-torps: 3</td>
</tr>
</tbody>
</table>

war-bird

<table>
<thead>
<tr>
<th>SHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos: (vec -10 10)</td>
</tr>
<tr>
<td>vel: (vec 10 0)</td>
</tr>
<tr>
<td>num-torps: 10</td>
</tr>
</tbody>
</table>
STAR WARS SIMULATOR

Environment Diagram

GE

make-ship: p1

P1

p: pos, vel, num-torps
b: (define move) ...
   (define fire-torp) ...
   ($ (msg)
      (cond ...))
(define enterprise
  (make-ship (make-vect 10 10)
    (make-vect 5 0)
    3))
(define enterprise
  (make-ship (make-vect 10 10)
              (make-vect 5 0)
              3))

(make-ship: p1

p: pos, vel, num-torps
b: (define move) ...
   (define fire-torp) ...
   (λ (msg)
      (cond ...))

pos: (vec 10 10)
vel: (vec 5 0)
num-torps: 3
move:
fire-torps:
(define enterprise
  (make-ship (make-vect 10 10)
              (make-vect 5 0)
              3))

stars wars simulator
environment diagram

(make-ship: p1)

(define move) ...
(define fire-torp) ...
(\lambda (msg)
  (cond ...)) |_{E_1}

pos: (vec 10 10)
vel: (vec 5 0)
num-torps: 3
move: p3
fire-torps:

p: pos, vel, num-torps
b: (define move) ...
  (define fire-torp) ...
  (\lambda (msg)
   (cond ...))

p: pos, vel, num-torps
b: (set! position ...)

E1

P3
(define enterprise
  (make-ship (make-vect 10 10)
    (make-vect 0 0)
    3))

STAR WARS SIMULATOR
Environment Diagram

GE

make-ship: p1

P1

p: pos, vel, num-torps
b: (define move) ...
  (define fire-torp) ...
    (λ (msg)
        (cond ...)) |_{E_1}

E1

pos: (vec 10 10)
vel: (vec 5 0)
num-torps: 3

move: p3
fire-torps: p4

P3

p:
b: (set! position ...)

E1

λ 

(define enterprise
  (make-ship (make-vect 10 10)
    (make-vect 5 0)
    3))

(make-ship: p1

P1

p: pos, vel, num-torps
b: (define move) ...
  (define fire-torp) ...
  (\(msg\)
    (cond ...) ) |E1

P2

p: msg
b: (cond ...)

P3

p: (set! position ...)

E1

pos: (vec 10 10)
vel: (vec 5 0)
um-torps: 3
move: p3
fire-torps: p4

GE
(define enterprise
  (make-ship (make-vect 10 10)
    (make-vect 5 0)
    3))
(define enterprise
  (make-ship (make-vect 10 10)
               (make-vect 5 0)
               3))

(enterprise 'MOVE) ; => done

GE

make-ship: \textit{p1}
enterprise: \textit{p2}

E1

pos: (vec 10 10)
vel: (vec 5 0)
um-torps: 3
move: \textit{p3}
fire-torps: \textit{p4}

E2

P1

p: pos, vel, num-torps
b: (define move) ...
  (define fire-torp) ...
  (\lambda (msg)
    (cond ...))

P2

p: msg
b: (cond ...)

P3

p:
b: (set! position ...)

P4

p:
b: (cond ...)

P5

p:
b: (cond ...)

\textit{E1}_1

\lambda (msg)
  (cond ...)}
(define enterprise
  (make-ship (make-vect 10 10)
             (make-vect 5 0)
             3))

(make-ship: p1
  enterprise: p2

(make-vect: p1
  pos: (vec 10 10)
  vel: (vec 5 0)
  num-torps: 3

(define move) ...
(define fire-torp) ...
(\(msg\) (cond ...)) \[E_1\]

(set! pos (add-vect pos vel)) \[E_2\]
pos \[E_2\] => (vec 10 10)
vel \[E_2\] => (vec 5 0)
(define enterprise
  (make-ship (make-vect 10 10)
    (make-vect 5 0)
    3))

(enterprise 'MOVE) ; => done
(enterprise 'POSITION) ; => (vect 15 10)
STAR WARS SIMULATOR
Extensions

• Add a `station` class to our world
• Add `predicate messages` to check type of objects
• Add `display` property to our system
  • `displays` the status of objects
  • can be implemented as a procedure
  • add `display` message to classes
    so that objects can `display` themselves upon request
    (by calling a `display` procedure)
STAR WARS SIMULATOR

Class Diagram

SHIP

position:
velocity:
um-torps:

position
velocity
move
attack
ship?
display

STATION

position:
position
station?
display
(define (make-station position)
  (lambda (msg)
    (cond ((eq? msg 'station?) #T)
          ((eq? msg 'position) position)
          ((eq? msg 'display)
             (newline)
             (display "station at ")
             (display position))
          (else (error "station can't " msg))))))
STAR WARS SIMULATOR

Further Extensions

• Animate our world!
  • Add a clock that moves time forward in the universe
  • Keep track of things (the *universe*)
  • Clock sends `clock-tick` message to objects
to have them update their state

• Add a torpedo class to system
STAR WARS SIMULATOR

Class Diagram

<table>
<thead>
<tr>
<th>TORPEDO</th>
<th>SHIP</th>
<th>STATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>position:</td>
<td>position:</td>
<td>position:</td>
</tr>
<tr>
<td>velocity:</td>
<td>velocity:</td>
<td>position</td>
</tr>
<tr>
<td>position</td>
<td></td>
<td>station?</td>
</tr>
<tr>
<td>velocity</td>
<td></td>
<td>display</td>
</tr>
<tr>
<td>move</td>
<td></td>
<td>explode</td>
</tr>
<tr>
<td>torpedo?</td>
<td></td>
<td>clock-tick</td>
</tr>
<tr>
<td>display</td>
<td></td>
<td></td>
</tr>
<tr>
<td>explode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clock-tick</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(define *universe* nil)

(define (add-to-universe thing)
  (set! *universe* (cons thing *universe*)))

(define (remove-from-universe thing)
  (set! *universe* (delq thing *universe*)))

STAR WARS SIMULATOR

*universe*
**STAR WARS SIMULATOR**

*universe*

SHIP
- **pos:** (vec 10 10)
- **vel:** (vec 5 0)
- **num-torps:** 3

SHIP
- **pos:** (vec -10 10)
- **vel:** (vec 10 0)
- **num-torps:** 10

TORPEDO
- **pos:** (vec -10 10)
- **vel:** (vec 10 0)

STATION
- **pos:** (vec -10 10)

enterprise

war-bird

earth
(define (clock)
  (for-each (lambda (x) (x 'clock-tick)) *universe*)
  (for-each (lambda (x) (x 'display)) *universe*)
  (let ((collisions (find-collisions *universe*)))
    (for-each (lambda (x) (x 'explode x))
      collisions)))
(define (clock)
  (for-each (lambda (x) (x 'clock-tick)) *universe*)
  (for-each (lambda (x) (x 'display)) *universe*)
  (let ((collisions (find-collisions *universe*)))
    (for-each (lambda (x) (x 'explode x))
      collisions)))

(define (run-clock n)
  (cond ((= n 0) 'done)
        (else (clock)
              (run-clock (- n 1)))))

{star wars simulator}

{tick-tock}
STAR WARS SIMULATOR

Update make-ship
STAR WARS SIMULATOR

implement make-torpedo
STAR WARS SIMULATOR

Implement find-collisions
STAR WARS SIMULATOR

Run a Simulation
SUMMARY
What have we learnt?

• Introduced a new programming style:
  • Object-oriented vs. Procedural
  • Uses – simulations, complex systems, ...

• Object-Oriented Modelling
  • Language independent!
    Class – template for state and behavior
    Instances – specific objects with their own identities

• Next time: powerful ideas of inheritance and delegation