CSC151.02 2013F, Class 50: Objects in Scheme

Overview

- Preliminaries.
  - Admin.
  - Questions on the project.
  - Questions on the exam.
- Motivating problems: Circles, turtles, and counters.
- Building and using compound values.
- Objects: A new approach to compound values.
- Creating objects in Scheme.
- Lab.

Preliminaries

Admin

- Quiz tomorrow, I think. - Match names to faces.
- Vote: Do you want Sam to talk about objects, or do you just want to do lab?
- Final lab writeup! Problem 6 (Due Friday)
  CSC151.02 Lab Writeup 31: Objects (Your Name Here)
- Some of you might find this interesting https://github.com/kyledreger/1000-nights
- BenGo sort:
- Upcoming extra credit opportunities:
  - Summer opportunities in Math 4:15 today. [Peer or Academic]
  - Any self-care week activity.
  - *One Grinnell* rally on December 4 at 4pm (unless you are taking photos).
  - Basketball Wednesday at 5:00.
  - Learning from Alumni Thursday at 2:15: Erik Hanson (in person)
  - CS Extras Thursday at 4:30: Summer Opportunities in CS
  - CS Table Friday: TBD.
  - Swim meet Friday/Saturday.

Questions on the Project

How do I tell if my work got uploaded?

No errors -> It probably worked. Sam will email you if it didn't.

Questions on the Exam

Motivating problems: Circles, turtles, and counters

One thing we do again and again and again is represent data

- An object that we want to display on the screen (shape, color, center, radius, ...)
- A student that we want to talk about (name, passtimes, ...)
- Turtle (brush, position, orientation, image, name, ...)

All of these group pieces together.

Some are mutable.

Building and using compound values

We want ways to represent thse within our programming language.

- We can use vectors. E.g., for turtles we might decide that
    - The brush goes in the position 0
    - Column in position 1
    - Row in position 2 *
- We could also use a list
- Why wouldn't we use a list? It may be more difficult to change.
- We could use a file.
- We could use a tree, althoug that's likely to be messy and complicated (and trees are as hard to modify)
- We could encode them in an integer.
- We could encode them in a string "brush,column,row,angle,image"
- We could use an association list, which means that we don't really care what order the things are stored in.
    - Slow to use
    - Hard to mutate

Simpler example: Counters:

- 1-element vector: the integer that keeps track of the count
- 2-elemetn vector: integer and the name

Code!

```
(define make-counter
  (lambda (name)
    (vector 0 name)))
(define display-counter
  (lambda (counter)
    (display (vector-ref counter 1))
    (display ": ")
    (display (vector-ref counter 0))
    (newline)))
(define increment-counter!
  (lambda (counter)
    (vector-set! counter 0 (+ 1 (vector-ref counter 0)))))

Welcome to DrRacket, version 5.2.1 [3m].
Language: racket; memory limit: 128 MB.
> (define sam (make-counter 'sam))
> (increment-counter! sam)
> (display-counter sam)
sam: 1
> (vector-set! sam 0 'samuel)
> (increment-counter! sam)
. . +: expects type <number> as 2nd argument, given: 'samuel; other arguments were: 1
```

Objects: A new approach to compound values

- Goal: Encapsulate the data and the operations on the data so that
  - People can use the data and operations
  - But only in the ways that we specify
- Some languages provide such encapsulation as a built-in command.

Creating objects in Scheme

Key idea: You can have functions with private data

```
(define sam
  (let ((x (vector 0 'sam)))
    (lambda ()
      (display (vector-ref x 1))
      (display ": ")
      (display (vector-ref x 0))
      (newline)
      (vector-set! x 0 (+ 1 (vector-ref x 0))))))

(define samr
  (let ((x (vector 0 'sam)))
    (lambda (command)
      (cond
        ((equal? command ':describe)
         (display (vector-ref x 1))
         (display ": ")
         (display (vector-ref x 0))
         (newline))
        ((equal? command ':increment!)
```

```
         (vector-set! x 0 (+ 1 (vector-ref x 0))))
        ((equal? command 'seuss)
         (display "I do like green eggs and ham."))
        (else
         (error "I do not know how to " command))))))
```

Note: The let/lambda combination creates a local variable that is accessible only within the procedure and that lives until the Scheme session ends.

Complexity: We don't want single objects, we want functions that build objects.

```
(define counter-new
  (lambda (name)
    (let ((vec (vector 0 name)))
      (lambda (command)
        (cond
          ((equal? command ':describe)
           (display (vector-ref vec 1))
           (display ": ")
           (display (vector-ref vec 0))
           (newline))
          ((equal? command ':increment!)
           (vector-set! vec 0 (+ 1 (vector-ref vec 0))))
          ((equal? command 'seuss)
           (display "I do like green eggs and ham."))
          (else
           (error "I do not know how to " command)))))))

> (define a (counter-new 'alpha))
> (define b (counter-new 'beta))
> (a ':describe)
alpha: 0
> (b ':increment!)
> (b ':increment!)
> (a ':describe)
alpha: 0
> (b ':describe)
beta: 2
> a
#<procedure>
> b
#<procedure>
>
```

Samuel A. Rebelsky, rebelsky@grinnell.edu

Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.