CS61A Notes – Week 7: Object oriented programming (solutions)
Taste the Rainbow (or: Dinner is not Ready)

```
(define-class (bag)
   (instance-vars (skittles '()))
   (method (tag-line) 'taste-the-rainbow)
   (method (add s) (set! skittles (cons s skittles)))
   (method (take) (let ((first (car skittles)))
                    (set! skittles (cdr skittles)) first))
   (method (take-color color)
        (let ((found (find (lambda(s) (eq? color (ask s 'color))) skittles)))
            (set! skittles (remove found skittles))
            found)))
```

Directories and Files (Again)

```
(define-class (file name content)
   (method (type) 'file)
   (method (size) (length content)))

(define-class (directory name)
   (instance-vars (content '()))
   (method (type) 'directory)
   (method (add thing) (set! content (cons thing content)))
   (method (mkdir dir) (ask self 'add (instantiate directory dir)))
   (method (cd dir) (find (lambda(f) (eq? dir (ask f 'name))) content))
   (method (mv thing dir)
        (let ((found (find (lambda(f) (eq? thing (ask f 'name)))
                           content)))
            (ask (ask self 'cd dir) 'add found)
            (set! content (remove found content))))
   (method (ls) (map (lambda(f) (ask f 'name)) content))
   (method (size)
        (accumulate (lambda(x y) (+ (ask x 'size) y)) 0 content)))
```

Pairs Are Objects Too

```
(define (make-oop-list ls)
   (cond ((null? ls) (instantiate the-null-list))
        (else (instantiate cons-pair (car ls) (make-oop-list (cdr ls))))))

(define-class (cons-pair the-car the-cdr)
   (method (length) (+ 1 (ask the-cdr 'length)))
   (method (listify) (cons the-car (ask the-cdr 'listify)))
   (method (map op)
        (instantiate (op the-car)
                     (ask the-cdr 'map op)))
   (method (map! op)
        (set! the-car (op the-car))
        (ask the-cdr 'map op))
   (method (accumulate comb init)
        (comb the-car (ask the-cdr 'accumulate comb init))))

(define-class (the-null-list)
   (method (listify) '())
   (method (map op) (instantiate the-null-list))
   (method (map! op) 'done) ;; return some dummy value
   (method (accumulate comb init) init))
```

Inheritance (or: Midterm Fun)

```
(define-class (question q a hint weight)
   (instance-vars (cur-answer '()))
   (method (read) q)
   (method (answer ans) (set! cur-answer ans))
   (method (grade) (if (equal? cur-answer a) weight 0))
   (method (hint pwd) ;; this overwrites the hint instantiation var.
           (if (equal? pwd 'redrum)
               hint
               '(Wrong password! I hope you are proud))))
   ;; note: is it a good idea to hard-code the password as 'redrum in
   ;; this case?  What are some advantages/disadvantages?  What if we
   ;; have password be an instantiation variable?

(define-class (bonus-question q)
   (parent (question q '() '(a bonus question gives no hints) 0)))
   ;; note: why don't we need to overwrite the grade method?

(define-class (midterm q-ls)
   (method (get-q n)
           (if (> n (- (length q-ls) 1))
               '(you are done)
               (list-ref q-ls n)))
   (method (grade)
           (accumulate (lambda(x y) (+ (ask x 'grade) y)) 0 q-ls)))

(define-class (proctor name)
   (method (answer msg) (append (list name ':) msg))
   (method (get-time) (random 100))
   (method (how-much-time-left?)
           (ask self 'answer (list (ask self 'get-time))))
   (method (clarify q) (ask self 'answer (ask q 'hint 'redrum))))

(define-class (instructor name)
   (parent (proctor name))
   (method (get-time) 30) ;; why didn't we overwrite how-much-time-left?
   (method (clarify q)
           (ask self 'answer '(the question is perfect as written))))

(define-class (ta name temper-limit)
   (parent (proctor name))
   (method (answer msg)
           (set! temper-limit (- temper-limit 1))
           (if (< temper-limit 0)
               (usual 'answer '(how the hell would I know?))
               (usual 'answer msg))))
   ;; note: when we (ask a-TA 'how-much-time-left?), it's going to
   ;; (ask self 'answer).  Which answer method will be called
   ;; (proctor's or TA's)?  Will only overwriting answer really work?

(define-class (lenient-proctor name p1 p2)
   (parent (proctor name))
   (method (get-time) ;; why not overwrite how-much-time-left?
           (max (ask p1 'get-time) (ask p2 'get-time))))
```