Quiz 2 Solutions

Problem 1:

```
(define (dept-name dept)
  (car dept))

(define (num-majors dept)
  (cadr dept))

(define (num-minors dept)
  (cddr dept))
```

Problem 2:

```
(define (list-of-majors school)
  (map dept-name school))
```

Problem 3:

```
(define (num-of-majors-in-school school)
  (accumulate + 0 (map num-majors school)))
```

Problem 4:

```
(define (change-major! school old-major new-major)
  (let ((old-dept (find-major old-major school))
         (new-dept (find-major new-major school)))
    (set-car! (dept-data old-dept) (- (num-majors old-dept) 1))
    (set-car! (dept-data new-dept) (+ (num-majors new-dept) 1))
    (list 'major 'changed 'from old-major 'to new-major!))

; you should return something from the procedure,
; rather than the value of a set!
Problem 5:

(define (make-department name num-majors num-minors)
  (let ((named-object (make-named-object name)))
    (lambda (message)
      (cond ((eq? message 'num-majors)
              (lambda (self) num-majors))
            ((eq? message 'num-minors)
              (lambda (self) num-minors))
            ((eq? message 'change-num-majors)
              (lambda (self new-num) (set! num-majors new-num) new-num))
            ((eq? message 'change-num-minors)
              (lambda (self new-num) (set! num-minors new-num) new-num))
            (else (get-method named-object message))))))

Problem 6: