Lab: Objects in Scheme

Summary: In this lab, we consider techniques for building objects, collections of data that support operations on those data.

Contents:

- Exercises
  - Exercise 1: Testing Switches
  - Exercise 2: A Multi-Stage Switch
  - Exercise 3: A Single Tally Object
  - Exercise 4: Making Generic Tallys
  - Exercise 5: Tallys with Initial Values
  - Exercise 6: Monitored Tallys

Exercises

Exercise 1: Testing Switches

a. Make a copy of the make-switch procedure from the reading.

b. Test the switches created by the make-switch procedure. Here are a few possible instructions.

```
> (define lamp-switch (make-switch))
> (define vacuum-cleaner-switch (make-switch))
> (lamp-switch ':show-position)
> (vacuum-cleaner-switch ':show-position)
> (lamp-switch ':toggle!)
> (lamp-switch ':show-position)
> (vacuum-cleaner-switch ':show-position)
> (lamp-switch ':toggle!)
> (vacuum-cleaner-switch ':toggle!)
> (lamp-switch ':show-position)
> (vacuum-cleaner-switch ':show-position)
```

Exercise 2: A Multi-Stage Switch

As you may know, some switches have more than two positions. For example, we might have a switch that switches between the values 0, 1, 2, and 3. (For a two-way light, 0 might represent “off”, 1 might represent “light one on”, 2 might represent “light two on”, and 3 might represent “both lights on”.) In general, you can only toggle to the next higher value (with 3 returning to 0 when toggled).

Implement four-stage-switch, which responds to the same messages as a switch, but with four possible stages.
Exercise 3: A Single Tally Object

Define a one-field object, `tally`, that responds to exactly four messages:

- `':show-contents`,
- `':set-contents-to-zero!`,
- `':increment!`, which has the effect of increasing the number stored in the `contents` field by 1.
- `':decrement!`, which has the effect of decreasing the number stored in the `contents` field by 1.

The initial value of the field should be 0.

For example,

```
> (tally ':set-contents-to-zero!)
(tally ':show-contents)
0
> (tally ':increment!)
(tally ':show-contents)
1
> (tally ':decrement!)
(tally ':decrement!)
(tally ':decrement!)
(tally ':show-contents)
-2
```

Note that you are creating a single object, not a procedure that creates objects.

Exercise 4: Making Generic Tallys

a. Define a `make-tally` procedure that constructs and returns objects similar to the `tally` object you defined in exercise 2.

b. Create two tally objects and demonstrate that they can be incremented and reset independently.

Exercise 5: Tallys with Initial Values

Write a new `make-tally` procedure that allows the client to create new tallys with a specified initial value. For example, I might say that a starting grade is 90 with

```
> (define grade (make-tally 90))
```

I would then increment and decrement it as students do good or bad work.

Exercise 6: Monitored Tallys

a. Define a constructor procedure, `make-monitored-tally`, for objects similar to the `tally` objects from exercise 2 above, except that each such object keeps track of the total number of messages that it has received.
Hint: For this exercise, you may want to make two vectors, one for the value of the tally and one for the count of operations. Alternately, you could make a two-element vector in which element 0 of that vector is the value of the tally and element 1 of that vector is the count of operations.

b. Test your procedure.