MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring 2006

Recitation 20 Solutions — 4/28/2006
Object-Oriented Systems II

Model Trains

Ben Bitdiddle has been hired as a consultant to analyze how long it takes passengers to travel on the T. Rather than try to analyze trains in a statistical sense, he decides that he's doing to construct a simulation, and see how long it takes for simulated riders to reach their destinations. Ben just finished working on his adventure game 6.001 project, so he uses the same framework.

He starts off with a `train` and a `station` class:

```
(define (train self name birthplace direction)
  (let ((person-part (person self name birthplace)))
    (make-handler
     'TRAIN
     (make-methods
      'END-OF-LINE
      (lambda ()
        (ask self 'SAY (list "arriving at"
                             (ask (ask self 'LOCATION) 'NAME)
                             "everybody off"))
        (set! direction
              (case direction
                ((east) 'west)
                ((west) 'east)
                ((north) 'south)
                ((south) 'north)))
        )
      'DIRECTION
      (lambda () direction)
      'TRAVEL
      (lambda ()
        (ask self 'GO direction)
        (and (ask (ask self 'LOCATION) 'IS-A 'STATION)
             (ask (ask self 'LOCATION) 'DESTINATION? direction)
             (ask self 'END-OF-LINE)))
      'INSTALL
      (lambda ()
        (ask our-clock
             'ADD-CALLBACK
             (create-clock-callback 'traintick self 'TRAVEL)))
      )
```

```
     person-part)))

(define (create-train name birthplace direction)
  (create-instance train name birthplace direction))

(define (station self name end-of-line)
  (let ((place-part (place self name)))
    (make-handler
     'STATION
     (make-methods
      'DESTINATION?
      (lambda (direction)
        (memq direction end-of-line)

      ))
     place-part)))

(define (create-station name end-of-lines)
  (create-instance station name end-of-lines))
```

Ben then goes on the define a `passenger` which will start out somewhere, wait for a train, board one when one arrives, and get off a the right place. He hasn't finished this one yet, but he wants to see how this works so far.

```
(define (passenger self name birthplace destination direction)
  (let ((person-part (person self name birthplace))
        (start-time #f)
        (state 'waiting))
    (make-handler
     'PASSENGER
     (make-methods
      'INSTALL
      (lambda ()
        (ask our-clock 'ADD-CALLBACK
             (create-clock-callback 'passtick self 'TRAVEL))
        (set! start-time (ask our-clock 'THE-TIME)))
      'TRAVEL
      (lambda ()
        (cond ((eq? state 'waiting)
               (ask self 'BOARD-TRAIN))
              ((eq? state 'arrived)
               'nothing)
              ((eq? state 'travelling)
               (ask self 'ARRIVED?))))
      'BOARD-TRAIN
      (lambda ()
        (let ((trains
```

```
               (filter (lambda (t) (and (ask t 'IS-A 'TRAIN)
                                        (eq? direction
                                             (ask t 'DIRECTION))))
                       (ask (ask self 'LOCATION)
                            'THINGS))))
          (if (not (null? trains))
              (begin
                (ask (car trains) 'ADD-THING self)
                (ask self 'SAY
                     (list "now boarding a train at"
                           (ask (ask self 'LOCATION) 'NAME)
                           "heading for"
                           (ask destination 'NAME)))
                (set! state 'travelling)))))
      'ARRIVED?
      (lambda ()
        (if (eq? (ask self 'LOCATION)
                 destination)
            (ask self 'SAY (list "arrived at"
                                 (ask (ask self 'LOCATION) 'NAME)))
            (ask self 'SAY (list "now at"
                                 (ask (ask self 'LOCATION) 'NAME)))))
      )
     person-part)))

(define (create-passenger name birthplace destination direction)
  (create-instance passenger name birthplace destination direction))
```

Ben then goes and creates a new `create-world` and `setup` procedure, which right now includes just a part of the red line:

```
(define (create-world)
  (let ((alewife (create-station 'alewife '(north)))
        (davis (create-station 'davis '()))
        (porter (create-station 'porter '()))
        (harvard (create-station 'harvard '()))
        (central (create-station 'central '()))
        (kendall (create-station 'kendall '()))
        (charles (create-station 'charles '()))
        (park (create-station 'park '()))
        (downtown-crossing (create-station 'downtown-crossing '()))
        (south-station (create-station 'south-station '(south))))
       ;add the rest of the line later


    (can-go-both-ways alewife 'south 'north davis)
    (can-go-both-ways davis 'south 'north porter)
```

```
    (can-go-both-ways porter 'south 'north harvard)
    (can-go-both-ways harvard 'south 'north central)
    (can-go-both-ways central 'south 'north kendall)
    (can-go-both-ways kendall 'south 'north charles)
    (can-go-both-ways charles 'south 'north park)
    (can-go-both-ways park 'south 'north downtown-crossing)
    (can-go-both-ways downtown-crossing 'south 'north south-station)

    (create-train 'red-line-1 south-station 'north)
    (create-train 'red-line-2 alewife 'south)

    (create-passenger 'alice kendall alewife 'north)
    (create-passenger 'bob harvard south-station 'south)

    (list alewife davis porter harvard central kendall charles park
          downtown-crossing south-station)))

(define (setup)
  (ask our-clock 'RESET)
  (display 'reset)
  (ask our-clock 'ADD-CALLBACK
       (create-clock-callback 'TICK-PRINTER our-clock 'PRINT-TICK))
  (display 'clock-setup)
  (let ((stations (create-world)))

       'ready))
```

1. Rather than writing all his code first, Ben tries out his model so far to see how it works:

```
(setup)
(run-clock 10)

--- THE-CLOCK Tick 0 ---
red-line-1 moves from south-station to downtown-crossing
red-line-2 moves from alewife to davis
--- THE-CLOCK Tick 1 ---
red-line-1 moves from downtown-crossing to park
red-line-2 moves from davis to porter
--- THE-CLOCK Tick 2 ---
red-line-1 moves from park to charles
red-line-2 moves from porter to harvard
At harvard bob says -- now boarding a train at harvard heading for south-station
--- THE-CLOCK Tick 3 ---
red-line-1 moves from charles to kendall
red-line-2 moves from harvard to central
At kendall alice says -- now boarding a train at kendall heading for alewife
At harvard bob says -- now at harvard
```

```
--- THE-CLOCK Tick 4 ---
red-line-1 moves from kendall to central
At central red-line-1 says -- Hi red-line-2
red-line-2 moves from central to kendall
At kendall alice says -- now at kendall
At harvard bob says -- now at harvard
--- THE-CLOCK Tick 5 ---
red-line-1 moves from central to harvard
red-line-2 moves from kendall to charles
At kendall alice says -- now at kendall
At harvard bob says -- now at harvard
--- THE-CLOCK Tick 6 ---
red-line-1 moves from harvard to porter
red-line-2 moves from charles to park
At kendall alice says -- now at kendall
At harvard bob says -- now at harvard
--- THE-CLOCK Tick 7 ---
red-line-1 moves from porter to davis
red-line-2 moves from park to downtown-crossing
At kendall alice says -- now at kendall
At harvard bob says -- now at harvard
--- THE-CLOCK Tick 8 ---
red-line-1 moves from davis to alewife
At alewife red-line-1 says -- arriving at alewife everybody off
red-line-2 moves from downtown-crossing to south-station
At south-station red-line-2 says -- arriving at south-station everybody off
At kendall alice says -- now at kendall
At harvard bob says -- now at harvard
--- THE-CLOCK Tick 9 ---
red-line-1 moves from alewife to davis
red-line-2 moves from south-station to downtown-crossing
At kendall alice says -- now at kendall
At harvard bob says -- now at harvard DONE
```

There's clearly something wrong here: Alice and Bob aren't moving along with their trains. What did Ben do wrong, and how should he fix it? (there's several different possible approaches)

The problem is that when the train moves, the location of all the people on the train doesn't change as well. The person's location will only change when the person has arrived at the destination.

Here's one possible solution, which just changes the passenger class:

```
(define (passenger self name birthplace destination direction)
  (let ((person-part (person self name birthplace))
        (start-time #f)
        (state 'waiting)
        (train #f))
```

```
(make-handler
 'PASSENGER
 (make-methods
  'INSTALL
  (lambda ()
    (ask our-clock 'ADD-CALLBACK
         (create-clock-callback 'passtick self 'TRAVEL))
    (set! start-time (ask our-clock 'THE-TIME)))
  'TRAVEL
  (lambda ()
    (cond ((eq? state 'waiting)
           (ask self 'BOARD-TRAIN))
          ((eq? state 'arrived)
           'nothing)
          ((eq? state 'travelling)
           (ask self 'ARRIVED?))))
  'BOARD-TRAIN
  (lambda ()
    (let ((trains
           (filter (lambda (t) (and (ask t 'IS-A 'TRAIN)
                                    (eq? direction
                                         (ask t 'DIRECTION))))
                   (ask (ask self 'LOCATION)
                        'THINGS))))
      (if (not (null? trains))
          (begin
            (ask (car trains) 'ADD-THING self)
            (set! train (car trains))
            (ask self 'SAY
                 (list "now boarding a train at"
                       (ask (ask self 'LOCATION) 'NAME)
                       "heading for"
                       (ask destination 'NAME)))
            (set! state 'travelling)))))
  'LOCATION
  (lambda ()
    (if (eq? state 'travelling)
        (ask train 'LOCATION)
        (ask person-part 'LOCATION)))
  'ARRIVED?
  (lambda ()
    (if (eq? (ask self 'LOCATION)
             destination)
        (begin (ask self 'SAY (list "arrived at"
                                    (ask (ask self 'LOCATION) 'NAME)))
               (ask self 'CHANGE-LOCATION (ask self 'LOCATION))
               (set! state 'arrived))
        (ask self 'SAY (list "now at"
```

```
                                    (ask (ask self 'LOCATION) 'NAME)))))
     )
    person-part)))
```

2. Right now, there's nothing preventing two trains from being on the same track at the same time, whereas in reality, if there's another train on the same track, the latter train will wait for the first one to move first. How might you go about simulating this behavior?

3. Ben's model for how trains work is rather simplistic. In reality, red line trains don't always take the same amount of time between stations, or stopped at each station – it depends on how many poeple are waiting at a station, how many other trains are on the same line, whether the Red Sox are in town, etc. Design some extensions to the system to make the travel time more realistic by modelling congestion in some way.

4. Simulation: Ben wants to know the total time passengers spend travelling. How would you run the simulation over and over again to determine what the possible trip times are?

5. Multiple lines. Not all passengers can get to their desintation on a single train – some have to go and switch trains at a middle station. How would you extend this system to handle such passengers?