OBJECT ORIENTED PROGRAMMING 12

GEORGE WANG
gswang.cs61a@gmail.com
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

July 13, 2010

1 Review

Things we talked about:

`define-class, instantiate, method, ask, instance-vars, instantiation variables, class-vars, parent, default-method`

2 Inheritance, Continued

What happens when you send an object a message for which there is no method defined in its class? If the class has no parent, this is an error. If the class does have a parent, and the parent class understands the message, it works as we've seen here. But you might want to create a class that follows some rule of your own devising for unknown messages:

```
;;;;; In file cs61a/lectures/3.0/demo2.scm
(define-class (squarer)
  (default-method (* message message))
  (method (7) 'buzz) )
> (define s (instantiate squarer))
> (ask s 6)       > (ask s 7)        > (ask s 8)
36                buzz               64
```

Within the default method, the name message refers to whatever message was sent. Let's say we want to maintain a list of all the instances that have been created in a certain class. It's easy enough to establish the list as a class variable, but we also have to make sure that each new instance automatically adds itself to the list. We do this with an INITIALIZE clause:

```
;;;;; In file cs61a/lectures/3.0/demo2.scm
(define-class (counter)
```

```
  (instance-vars (count 0))
  (class-vars (total 0) (counters '()))
  (initialize (set! counters (cons self counters)))
  (method (next) (set! total (+ total 1))
          (set! count (+ count 1))
          (list count total)))
> (define c1 (instantiate counter))
> (define c2 (instantiate counter))
> (ask counter 'counters)
(#<procedure> #<procedure>)
```

There was a bug in our pigger class definition; Scheme gets into an infinite loop if we ask Porky to greet, because it tries to translate the word my into Pig Latin but there are no vowels aeiou in that word. To get around this problem, we can redefine the pigger class so that its say method says every word in Pig Latin except for the word my, which it'll say using the USUAL method that persons who aren't piggers use:

```
;;;;; In file cs61a/lectures/3.0/demo2.scm
(define-class (pigger name)
  (parent (person name))
  (method (pigl wd)
    (if (member? (first wd) '(a e i o u))
        (word wd 'ay)
        (ask self 'pigl
                  (word (bf wd) (first wd))) ))
  (method (say stuff)
    (if (atom? stuff)
        (if (equal? stuff 'my)
            (usual 'say stuff)
            (ask self 'pigl stuff))
        (map (lambda (w) (ask self 'say w)) stuff))) )
> (define porky (instantiate pigger 'porky))
> (ask porky 'greet)
(ellohay my amenay isay orkypay)
```

(Notice that we had to create a new instance of the new class. Just doing a new define-class doesn't change any instances that have already been created in the old class. Watch out for this while you're debugging the OOP programming project.) We invoke usual in the say method to mean "say this stuff in the usual way, the way that my parent class would use."

The last thing we talk about is multiple inheritance. This is the idea that children can have multiple parents. The way to think about how the parents are 'picked' to evaluate what needs to happen next is recursive search. You know this to be Depth First Search. In other words, we pick the first parent, and recursively apply that rule, until we get to the second parent, or a method that handles it, and we repeat this over and over.