Reminder

- Start thinking of ideas of PS9 and discussing them on the forum
- You can also vote in the "should we have a quiz Monday" poll
- http://www.sportsline.com/collegebasketball/scoreboard

Problem-Solving Strategies

- PS1-PS4: Functional Programming
  - Focused on **procedures**
  - Break a problem into procedures that can be combined to solve it
- PS5: Imperative Programming
  - Focused on **data**
  - Design data for representing a problem and procedures for updating that data

Counter Object

```
(define (make-counter)
  (let ((count 0))
    (lambda (message)
      (cond ((eq? message 'reset!)
             (set! count 0))
             ((eq? message 'next!)
              (set! count (+ 1 count)))
             ((eq? message 'current)
              count)
             (else
              (error "Unrecognized message"))))))
```

Problem-Solving Strategies

- PS6: “Object-Oriented Programming”
  - Focused on **objects**: package procedures and state
  - Model a problem by dividing it into objects
  - Lots of problems in real (and imaginary) worlds can be thought of this way

```
(define (ask object message)
  (object message))
```

Defining **ask**

```
> (define bcounter (make-counter))
> (ask bcounter 'current)
0
> (ask bcounter 'next)
1
```

```
(define (ask object message)
  (object message))
```
Lecture 23: Programming with Objects

Inheritance

There are many kinds of numbers...

- Whole Numbers (0, 1, 2, ...)
- Integers (-23, 73, 0, ...)
- Fractions (1/2, 7/8, ...)
- Floating Point (2.3, 0.0004, 3.14159)

- But they can’t all do the same things
  - We can get the denominator of a fraction, but not of an integer

make-fraction

(define make-fraction
  (lambda (numerator denominator)
    (lambda (message)
      (cond
        ((eq? message 'value)
         (lambda (self) (/ numerator denominator)))
        ((eq? message 'add)
         (lambda (self other)
           (+ (ask self 'value) (ask other 'value)))
        ((eq? message 'get-numerator)
         (lambda (self) numerator))
        ((eq? message 'get-denominator)
         (lambda (self) denominator))
        (else
         (super message))))))

Why is redefining add a bad thing?

- Cut-and-paste is easy but...
- There could be lots of number methods (subtract, multiply, print, etc.)
- Making the code bigger makes it harder to understand
- If we fix a problem in the number add method, we have to remember to fix the copy in make-fraction also (and real, complex, float, etc.)

Making Subobjects

(define (make-fraction numer denom)
  (let ((super (make-number #f)))
    (lambda (message)
      (cond
        ((eq? message 'value)
         (lambda (self) (/ numer denom)))
        ((eq? message 'get-denominator)
         (lambda (self) denom))
        ((eq? message 'get-numerator)
         (lambda (self) numer))
        (else
         (super message))))))

Note: our add method evaluates to a number, not a fraction object (which would be better).
Implementing make-subobject

(define (make-subobject super imp)
  (lambda (message)
    (if (eq? message 'super)
        (lambda (self) super)
        (let ((method (imp message)))
          (if method
              method
              (super message))))))

Using Fractions

> (define half (make-fraction 1 2))
> (ask half 'value)
1/2
> (ask half 'get-denominator)
2
> (ask half 'add (make-number 1))
3/2
> (ask half 'add half)
1

(make-number)
(make-fraction)

Inheritance

Inheritance is using the definition of one class to make another class

make-fraction uses make-number to inherit the behaviors of number

Speaking about Inheritance

Fraction inherits from Number.

Fraction is a subclass of Number.

The superclass of Fraction is Number.
PS6

Make an adventure game programming with objects

Many objects in our game have similar properties and behaviors, so we use inheritance.

**PS6 Classes**

sim-object

physical-object — place

mobile-object — thing

person — student — police-officer

**make-class** is the procedure for constructing objects in the class class

student inherits from person which inherits from mobile-object which inherits from physical-object which inherits from sim-object.

**Charge**

- Monday:
  - Quiz on GEB reading (depending on poll)
  - History of Object-Oriented Programming
- PS6 due Friday
- Start thinking about PS9 project ideas
  - Use the forum to find teammates and propose ideas

**Are there class hierarchies like this in the “real world” or just in fictional worlds like Charlottansville?**