# Dan Grossman, CSE341, Programming Languages  
# Lecture 22, Stage A: OOP vs. Functional Decomposition

Note: If Exp and Value are empty classes, we do not need them in a  
dynamically typed language, but they help show the structure and they  
can be useful places for code that applies to multiple subclasses.

class Exp  
  # could put default implementations or helper methods here
end

class Value < Exp  
  # this is overkill here, but is useful if you have multiple kinds of  
  # /values/ in your language that can share methods that do not make sense  
  # for non-value expressions
  
  def eval  
    # no argument because no environment
    self
  end
end

class Int < Value  
  attr_reader :i
  def initialize i
    @i = i
  end
  def toString
    @i.to_s
  end
  def hasZero
    i==0
  end
end

class Negate < Exp  
  attr_reader :e
  def initialize e
    @e = e
  end
  def eval
    Int.new(-e.eval.i)  # error if e.eval has no i method
  end
  def toString
    "- (" + e.toString + ")"
end
  def hasZero
    e.hasZero
  end
end

class Add < Exp  
  attr_reader :e1, :e2
  def initialize(e1,e2)
    @e1 = e1
    @e2 = e2
  end
  def eval
    Int.new(e1.eval.i + e2.eval.i)  # error if e1.eval or e2.eval has no i method
  end
  def toString
    "(" + e1.toString + " + " + e2.toString + ")"
end
  def hasZero
    e1.hasZero || e2.hasZero
  end
end