# Mixins
module Doubler
    def double
        self + self
    end
end

# simple 2D point class that includes the Doubler Mixin
# Note: This class provides an implementation of +
class Pt
    attr_accessor :x, :y
    include Doubler
    def + other
        ans = Pt.new
        ans.x = self.x + other.x
        ans.y = self.y + other.y
        ans
    end
end

# Questionable style but still interesting...
class Fixnum
    include Doubler
end

class String
    include Doubler
end

class Name
    attr_accessor :first, :middle, :last
    include Comparable
    def initialize(first, last, middle="")
        @first = first
        @last = last
        @middle = middle
    end
    def <=> other
        l = @last <=> other.last # <=> defined on strings
        return l if l != 0
        f = @first <=> other.first
        return f if f != 0
        @middle <=> other.middle
    end
end

# Using the Comparable Mixin
# Note: This class provides an implementation of <=>
class Name
    attr_accessor :first, :middle, :last
    include Comparable
    def initialize(first, last, middle="")
        @first = first
        @last = last
        @middle = middle
    end
    def <=> other
        l = @last <=> other.last # <=> defined on strings
        return l if l != 0
        f = @first <=> other.first
        return f if f != 0
        @middle <=> other.middle
    end
end

# The Enumerable Mixin
# Simple BinaryTree with insert and lookup methods
# Note: This class provides an implementation of each
class BinaryTree
    class Node
        attr_accessor :left, :right
        attr_reader :data
        def initialize(data, left=nil, right=nil)
            @data = data
            @left = left
            @right = right
        end
    end
end

# Let's add 47 other awesome methods
include Enumerable

def insert el
    @root = insert_(el, @root)
end

def lookup el
    lookup_(el, @root)
end

def each
    # Note the "necessary block wrapping"
each_(@root) { |x| yield x }
end

private
def insert_ el, root
    return Node.new(el) unless root
    if el < root.data
        root.left = insert_(el, root.left)
    else
        root.right = insert_(el, root.right)
    end
    root
end
def lookup_ el, root
    return el if root.data == el
    if el < root.data
        lookup_(el, root.left)
    else
        lookup_(el, root.right)
    end
end
def each_ root
    if root
        each_(root.left) { |x| yield x }
        yield root.data
        each_(root.right) { |x| yield x }
    end
end