class Pt
  attr_accessor :x, :y
  def distToOrigin
    Math.sqrt(x * x + y * y)
  end
end

class ColorPt < Pt
  attr_accessor :color
  def darken # error if @color not already set
    self.color = "dark" + self.color
  end
end

class Pt3D < Pt
  attr_accessor :z
  def distToOrigin
    Math.sqrt(x * x + y * y + z * z)
  end
end

# This does not exist in Ruby (or Java/C#, it does in C++)
# class ColorPt3D_3 < ColorPt, Pt3D
# end
# two ways we could actually make 3D Color Points:
class ColorPt3D_1 < ColorPt
  attr_accessor :z
  def distToOrigin
    Math.sqrt(x * x + y * y + z * z)
  end
end

class ColorPt3D_2 < Pt3D
  attr_accessor :color
  def darken
    self.color = "dark" + self.color
  end
end

############# mixins #############
module Doubler
  def double
    self + self # uses self’s + message, not defined in Doubler
  end
end

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

class String
  include Doubler
end

# these are probably the two most common uses in the Ruby library:
# Comparable and Enumerable
# you define <=> and you get ==, >, <, >=, <= from the mixin
# (overrides Object’s ==, adds the others)
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
    return l if l != 0
    f = @first <=> other.first
    return f if f != 0
    @middle <=> other.middle
  end
end

# Note ranges are built in and very common
# you define each and you get map, any?, etc.
# (note map returns an array though)
class MyRange
  include Enumerable
  def initialize(low,high)
    @low = low
    @high = high
  end
  def each
    i=@low
    while i <= @high
      yield i
      i=i+1
    end
  end
end

# here is how module Enumerable could implement map:
# (but notice Enumerable’s map returns an array,
# *not* another instance of the class :(
# def map
#   arr = []
#   each do |x| arr.push x
#   end
# end

# this is more questionable style because the mixin is using an
# instance variable that could clash with classes and has to be initialized
module Color
  def color
    @color
  end
  def color= c
    @color = c
  end
  def darken
    self.color = "dark" + self.color
  end
end

class Pt3D_again < Pt
  attr_accessor :z
  include Color
end

class ColorPt_again < Pt
  include Color
end
end

class ColorPt3D_again < Pt3D
  include Color
end