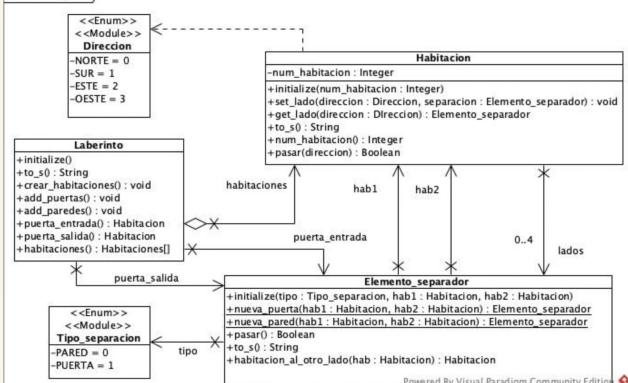


### Class Diagram 1



### Laberinto.rb

require\_relative "habitacion.rb"  
require\_relative "elemento\_separador.rb" } relaciones de asociación

module Modulo\_Laberinto → importado en el sistema de archivos.

class Laberinto

attr\_reader :puerta\_entrada, :puerta\_salida, :habitaciones  
getter implícito de las roles.  
(yo que en el diagrama de clases aparecen como públicos).

} Constructor.

def initialize  
@puerta\_entrada = nil  
@puerta\_salida = nil  
@habitaciones = []  
@paredes = []  
@puertas = []  
end

def crear\_habitaciones  
@habitaciones = Array.new(5)  
5.times do |index|  
 @habitaciones[index] = Habitacion.new(index+1)  
 for direccion in 0..3  
 @habitaciones[index].set\_lado(direccion, nil) } los puertos de todas las habitaciones serán nil.  
 end  
end

def add\_paredes  
pared = Elemento\_separador.nueva\_pared(@habitaciones[0], nil)  
@habitaciones[0].set\_lado(Direccion::norte, pared)  
pared = Elemento\_separador.nueva\_pared(@habitaciones[0], nil)  
@habitaciones[0].set\_lado(Direccion::este, pared)  
pared = Elemento\_separador.nueva\_pared(@habitaciones[0], nil)  
@habitaciones[0].set\_lado(Direccion::sur, pared)  
pared = Elemento\_separador.nueva\_pared(@habitaciones[0], nil)  
@habitaciones[0].set\_lado(Direccion::este, pared)  
pared = Elemento\_separador.nueva\_pared(@habitaciones[0], nil)  
@habitaciones[0].set\_lado(Direccion::sur, pared)

pared = Elemento\_separador.nueva\_pared(@habitaciones[1], nil)  
@habitaciones[1].set\_lado(Direccion::sur, pared)  
pared = Elemento\_separador.nueva\_pared(@habitaciones[1], nil)  
@habitaciones[1].set\_lado(Direccion::este, pared)

pared = Elemento\_separador.nueva\_pared(@habitaciones[2], nil)  
@habitaciones[2].set\_lado(Direccion::este, pared)  
pared = Elemento\_separador.nueva\_pared(@habitaciones[2], nil)  
@habitaciones[2].set\_lado(Direccion::sur, pared)

pared = Elemento\_separador.nueva\_pared(@habitaciones[3], nil)  
@habitaciones[3].set\_lado(Direccion::sur, pared)  
pared = Elemento\_separador.nueva\_pared(@habitaciones[3], nil)  
@habitaciones[3].set\_lado(Direccion::este, pared)

pared = Elemento\_separador.nueva\_pared(@habitaciones[4], nil)  
@habitaciones[4].set\_lado(Direccion::este, pared)  
pared = Elemento\_separador.nueva\_pared(@habitaciones[4], nil)  
@habitaciones[4].set\_lado(Direccion::sur, pared)

end

def add\_puertas  
puerto = Elemento\_separador.nueva\_puerta(@habitaciones[0], @habitaciones[1])  
@habitaciones[0].set\_lado(Direccion::ESTE, puerto) } la puerta de una será  
@habitaciones[1].set\_lado(Direccion::OESTE, puerto) } la opuesta de la otra.  
puerto = Elemento\_separador.nueva\_puerta(@habitaciones[1], @habitaciones[2])  
@habitaciones[1].set\_lado(Direccion::ESTE, puerto)  
puerto = Elemento\_separador.nueva\_puerta(@habitaciones[2], @habitaciones[3])  
@habitaciones[2].set\_lado(Direccion::ESTE, puerto)

@puerta\_entrada = Elemento\_separador.nueva\_puerta(@habitaciones[0], nil)  
@habitaciones[0].set\_lado(Direccion::sur, @puerta\_entrada)  
@puerta\_salida = Elemento\_separador.nueva\_puerta(@habitaciones[4], nil)  
@habitaciones[4].set\_lado(Direccion::norte, @puerta\_salida)

end

def to\_s  
s = "Habitaciones del laberinto:\n" } Equivalente al método  
s += @habitaciones.join  
end

} join() → une todos los elementos  
de un array o string con un separador  
que se pasa por parámetro  
→ Método del lenguaje.

### habitacion.rb

require\_relative "direccion.rb" → existe que se hemos incluido con require\_relative o comando require.  
module Modulo\_Laberinto  
A prior de que en el código se establezca  
una relación de asociación.

attr\_reader :num\_habitacion

def initialize(num\_habitacion)  
 @num\_habitacion = num\_habitacion  
 @lados = Array.new(4)  
end

def set\_lados(direccion, separador)  
 @lados[direccion] = separador  
end

def get\_lado(direccion)  
 @lados[direccion]  
end

def pasar(direccion)  
 if get\_lado(direccion) == nil || get\_lado(direccion).tipo == Tipo\_separacion::PUERTA  
 return true  
 else return false  
 end  
end

def to\_s  
 p "Hab " + @num\_habitacion.to\_s + " tiene los lados:\n" } para imprimir  
 p " " + @lados[Direccion::norte].to\_s + "\n" } para imprimir  
 p " " + @lados[Direccion::sur].to\_s + "\n" } para imprimir  
 p " " + @lados[Direccion::este].to\_s + "\n" } para imprimir  
 p " " + @lados[Direccion::oeste].to\_s + "\n" } para imprimir  
end

### elemento\_separador.rb

require\_relative "tipo\_separacion.rb"  
require\_relative "habitacion.rb"  
module Modulo\_Laberinto  
class Elemento\_separador  
attr\_reader :tipos, :hab1, :hab2  
def initialize(tipo, hab1, hab2)  
 @tipos = tipo  
 @hab1 = hab1  
 @hab2 = hab2  
end

private\_class\_method :new → tiene de especificar que un método de clase es privado.  
def new(num, puerta, hab1, hab2)  
 new(Tipo\_separacion::PUERTA, puerta, hab1, hab2)  
end

def self.num\_puerta(hab1, hab2)  
 new(Tipo\_separacion::PUERTA, hab1, hab2)  
end

def habitacion\_en\_otro\_lado(hab)  
 if @hab1 == hab  
 then return @hab2  
 else return @hab1 → se entiende bien que se hace aquí:  
 end  
end

def pasar  
 @tipos == Tipo\_separacion::PUERTA  
end

def to\_s  
 if @tipos == Tipo\_separacion::PUERTA  
 if @hab1 != nil  
 s = "S/ una puerta entre " + @hab1.num\_habitacion.to\_s + " y " + @hab2.num\_habitacion.to\_s  
 else  
 s = "S/ una puerta de " + @hab1.num\_habitacion.to\_s + " al exterior"  
 end  
 else  
 if @hab2 != nil  
 s = "S/ una pared entre " + @hab1.num\_habitacion.to\_s + " y " + @hab2.num\_habitacion.to\_s  
 else  
 s = "S/ una pared entre " + @hab1.num\_habitacion.to\_s + " al exterior"  
 end  
 end  
end