# Metaprogramación

## Objetos 3

# Self - Modification

# Self modification

---

Capacidad que provee el lenguaje y en particular la API de metaprogramación para no sólo “reflejar” el metamodelo, sino para modificarlo.

Ejemplos

- Agregar métodos
- Agregar variables de instancia / clase
- Modificar las relaciones de herencia
- etc

# Self - Modification - Temas

---

- Modificando clases - ejemplos básicos
- Mixins & Metaprogramación (extend & include)
- Patrones y diseño usando self-modification
- Method missing

# Self Modifications - Uso básico

---

Agregar una variable de instancia

No hay un método específico, porque en Ruby no se declaran las variables de instancia. Simplemente se definen.

```
p.instance_variable_set(:@nombre, nil)

p.instance_variable_get(:@nombre)
```

# Self Modification - Agregar métodos (1)

---

Agregar un método. Con “define_method”. Pero es privado :(

```
Persona.send(:define_method, :nombre) { @nombre }

Persona.send(:define_method, :nombre=) { |arg |@nombre = arg
}

p.nombre = ‘pepe’

p.nombre  // pepe
```

# Self Modification - Agregar métodos (2) - OpenClasses

---

Usamos el concepto de OpenClasses (volvemos a declarar la misma clase)

```
class Persona
    def apellido
      @apellido
    end
    def apellido=(nuevo)
      @apellido = nuevo
    end
end
```

# Self Modification - Agregar métodos (3) - “Def”

---

El “def” que ya usabamos se puede utilizar. Pero ojo ! a quién agrega el comportamiento ?

```
def Persona.saludar; “ALOHA” ; end

p.saludar  // FALLA ! No existe !
```

Porque el método se agregó al objeto Persona, o sea, a la clase.

```
Persona.saludar  // FUNCIONA !
```

# Self Modification - Agregar métodos (3) - “Def”

---

Se puede usar el def para agregar a una instancia (y sólo a esa instancia!)

```
def p.nombreCompleto; @nombre + " " + @apellido; end


p.nombreCompleto

>> "pepe Argento"

otro = Persona.new

otro.nombreCompleto // FALLA !!
```

# Self Modification - Agregar métodos

Resumen

- Hay varias formas de agregar métodos.
- Se pueden agregar métodos a instancias particulares.
  - Poderoso
  - “complejo” (si quiero definir comportamiento para todas las instancias no me sirve, tengo que pensar otro mecanismo)
- Para agregar “métodos de instancia” a una clase y que apliquen a todas las instancias:
  - send(“define_method”):  imperativo, pero más flexible
  - OpenClass: declarativo, pero rígido.

# Mixins & Metaprogramación

# Extend & Include

---

- Existen dos métodos que sirven para “incrustar” métodos de un mixin (module).
- Ambos se envían a un objeto de tipo Class.
- **include**
  - para agregar **métodos de instancia.**
  - Es el mismo “include” que usabamos para combinar un mixin en una clase
- **extend**
  - para **agregar los métodos como métodos de clase**

# Include: métodos de instancia

---

```
module Piloto
  def volar(avion)
    "vuelo en un " + avion
  end
end
```

```
Persona.include(Piloto)
```

# Include: métodos de instancia (...)

```
---
>> p.volar("SESNA")
=> "vuelo en un SESNA"
>> otro.volar("MIG")
=> "vuelo en un MIG"
```

# extend: métodos de clase

– – –

```
module Aviones
    def sesna
      'Sesna'
    end
    def mig
      'MIG'
    end
end
```

# extend: métodos de clase (...)

---

```
>> Persona.extend(Aviones)
>> otro.sesna // NoMethodError, método de clase!
>> Persona.sesna
=> “Sesna”
```

# Patrones

# Patrones de uso de SelfModification con Mixins

---

- Para agregar **métodos de instancia y métodos de clase**
- Usamos un Module/Mixin
- Tienen un “hook” cuando son combinados en una clase
- Usamos ese hook para
  - Agregar métodos de instancia con “include” (de otro mixin)
  - Agregar métodos de clase con extend (de otro mixin)

# Patrones de uso de SelfModification con Mixins

---

```
module MiFeature

 module MetodosInstancia
   def saludar
     'hola'
   end
 end

...
```

# Patrones de uso de SelfModification con Mixins

---

```
...

module MetodosDeClase
  def despedir
    "Chau, soy un método de clase"
  end
end

...
```

# Patrones de uso de SelfModification con Mixins

---

```
 ...

   def self.included(base)
       base.include(MetodosInstancia)
       base.extend(MetodosDeClase)
   end

end
```

# Patrones de uso de Self-Modification con Mixins

---

```
class MiClase
 include MiFeature
end

MiClase.despedir
=> "Chau, soy un método de clase"

MiClase.new.saludar()
=> "hola"
```

# Method - Missing

# Method Missing

---

- Soporte del lenguaje para intervenir en el method dispatch.
- “hook” cuando un objeto no entiende un mensaje
- Poderoso !

# Method Missing

---

```ruby
class EntiendeTodo
 def saludar
   "Hola"
 end

 def method_missing(symbol, *args)
   "No se #{symbol}(#{args})"
 end
end
```

# Method Missing - “Delegates”

– – –

Ejemplo más elaborado

- Una clase puede declarar 1 o más objetos “delegados”
- Cuando una instancia no entiende el mensaje que le envían lo delega en estos objetos
- El orden importa.

# Method Missing - “Delegates”

– – –

Ejemplo de uso

```
class CalculadorDePrecios
  include Prototipado

  delegate Precios.new

  def promedio
      (bebidas + comidas) / 2
  end
end
```

```
class Precios
  def bebidas
      30
  end

  def comidas
      12
  end
end
```

# Extras

---

- instance_eval: sirve para evaluar un bloque definiendole cual va a ser el objeto referenciado por “self”.