IIC2513 - TECNOLOGÍAS Y APLICACIONES WEB

I-2016

RUBY - PARTE 2

SÍMBOLOS vs STRINGS

“anaconda”

“type1”

“pokemon”

“Hola mundo”

“jugar”

“Pikachu”

“persona”

“amateurismo”

“anda”

“foo”

“Retrotraer”

“Había una vez … truz”

“Beatriz”

“find”

“quitasol”

“OMG”

“chiste”

“catalog”

“pokemon”

“persona”

“Pikachu”

“Había una vez … truz”

“Beatriz”

“chiste”

“persona”

“Beatriz”

“pokemon”

“Pikachu”

“chiste”

“Había una vez … truz”

```
cosas = {
  “persona” => “Beatriz”,
  “pokemon” => “Pikachu”,
  “chiste” => “Había una vez… truz”
}
```

```
puts cosas[“persona”]
unless cosas.has?(“pokemon”)
  puts “:( no tengo”
end
# ...
```

Nota: obviamente string reales no pueden ser con comillas tipográficas: “”

Símbolos

(clase Symbol)

Símbolo

- Es como un String.
- Pero es más
- Como si un String
- Se hubiera casado con una Constante
- El String y la Constante tuvieron hijos

Símbolo

- Los hijos se fueron a vivir a una isla
- Crearon una sociedad propia
- La sociedad colapsó
- Murieron
- Revivieron como un fénix

Símbolo

- Ese fénix

Símbolo

Es como un string, pero constante

“string”

“otro string”

:simbolo

: "otro simbolo"

Símbolo

- Nunca se eliminan de memoria
- Muy eficientes
- Es como una constante con nombre
- Cuyo valor es si misma

USO DE SÍMBOLOS

```
cosas = {
  “persona” => “Beatriz”,
  “pokemon” => “Pikachu”,
  “chiste” => “Había una vez… truz”
}
```

```
puts cosas[“persona”]
unless cosas.has?(“pokemon”)
  puts “:( no tengo”
end
# ...
```

Nota: obviamente string reales no pueden ser con comillas tipográficas: “”

```
cosas = {
  :persona => “Beatriz”,
  :pokemon => “Pikachu”,
  :chiste => “Había una vez… truz”
}
```

```
puts cosas[:persona]
unless cosas.has?(:pokemon)
  puts “:( no tengo”
end
# ...
```

Nota: obviamente string reales no pueden ser con comillas tipográficas: “”

```
cosas = {
  :persona => “Beatriz”,
  :pokemon => “Pikachu”,
  :chiste => “Había una vez… truz”
}
```

```
pokemon:  “Pik
```

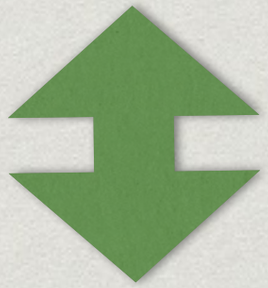
```
cosas = {
  :persona => “Beatriz”,
  :pokemon => “Pikachu”,
  :chiste => “Había una vez… truz”
}
```

Nota: obviamente string reales no pueden ser con comillas tipográficas: “”

```
cosas = {
  persona: “Beatriz”,
  pokemon: “Pikachu”,
  chiste: “Había una vez… truz”
}
```

Nota: obviamente string reales no pueden ser con comillas tipográficas: “”

ble:



:ble =>

HASHES

HASHES EVERYWHERE

MÉTODOS

nombre método (

```
    param1 ,
    param2 ,
    … ,
    paramk ,
```

hash) bloque

formatear_moneda (:uf, 300.5)

formatear_moneda (:uf, 300.5, {:decimales => 2, :simbolo => “UF”})

formatear_moneda (:uf, 300.5, :decimales => 2, :simbolo => “UF”)

formatear_moneda (:uf, 300.5, decimales: 2, simbolo: “UF”)

formatear_moneda (:uf, 300.5, decimales: 2, simbolo: “UF”)

Ejemplo

```
<input type="text" id="post_title" name="post[title]" size="20" value="#{@post.title}" />
```

text_field(object_name, method, options = {})

```
text_field(:post, :title, :size => 20)
```

```
text_field(:post, :title)
```

```
text_field(:snippet, :code, :size => 20, :class => 'code_input')
```

BLOQUES

Es un trozo de código

```ruby
[2,7,1,4].each do |value|
  puts value
end
```

```ruby
dogs = Dog.all # arreglo con todos los perros

# perros a razas
races = dogs.map{|dog| dog.race}

# obviamente estaran repetidas
races.uniq!
```

```
metodo() do |param1, param2, ...|
  # código
  # de
  # muchas
  # lineas
end
```

```
metodo() { |param1, ...|  "código de una linea" }
```

¿Cómo hago un método que use un bloque?

```ruby
# inspirado en http://slipsum.com/
def slipsum_wrapper()
  text = "MOTHERFUCKING "

  block_result = ""
  if block_given?
    block_result = yield
  end
  text += block_result.to_s

  text += " MOTHERFUCKER!"

  text
end

hello = slipsum_wrapper { "hi" }

puts hello
# MOTHERFUCKING hi MOTHERFUCKER!
```

OTRAS COSAS ÚTILES

try

send

send

Permite llamar un método

Ejemplos:

```
my_array.send(:push, 5)

my_array.send(:reduce, 0, :+) # my_array.reduce(0,:+)

some_object.send(“find_by_#{var}”)
```

try

Recibe el “nombre” de un método y (opcionalmente), parámetros.

Si el objeto es **nil**, retorna **nil**.

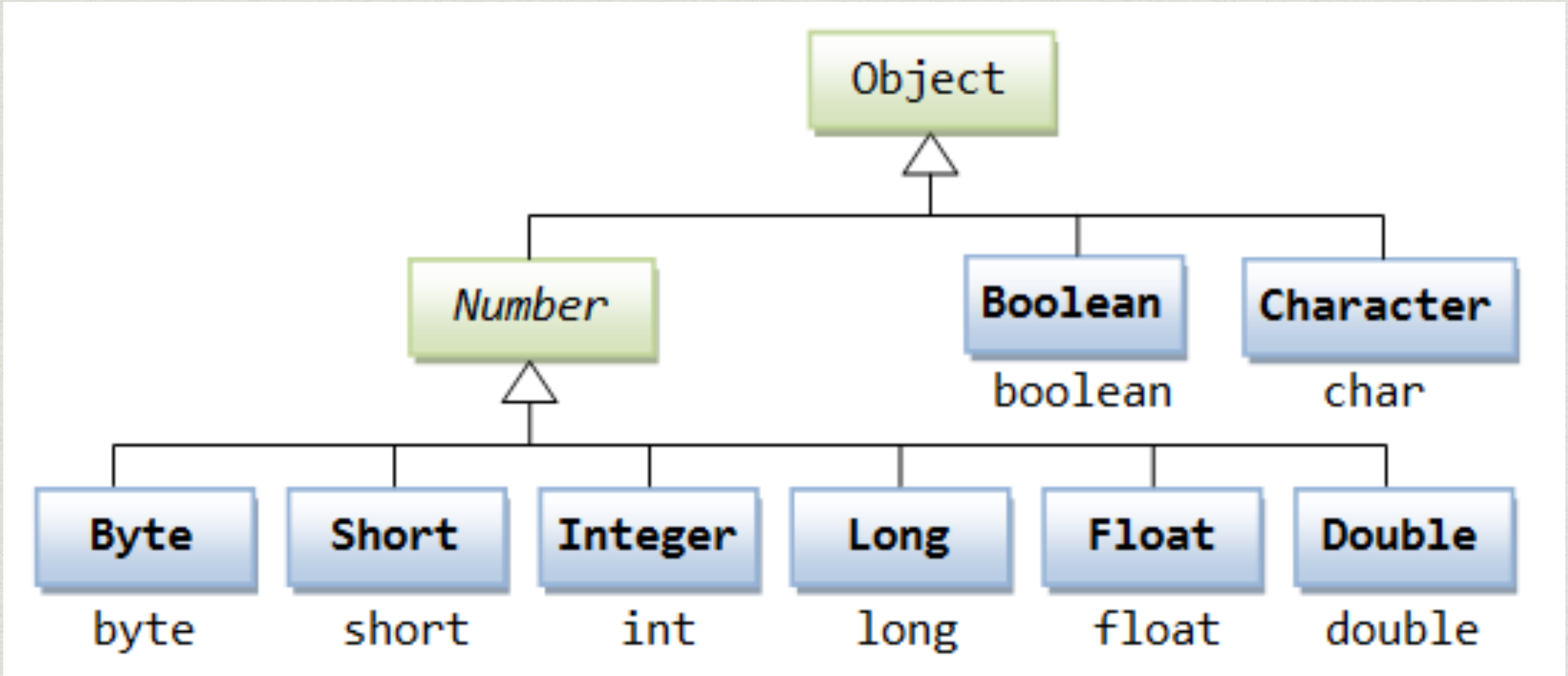
Sino, llama al método y entrega el valor de retorno.

Ejemplos:

```
tag.try(:append, “div”, id: ‘the_child’, class: ‘no-border’)

num.try(:+, 2)
```

JERARQUÍA DE CLASES



“tipos primitivos” en Java

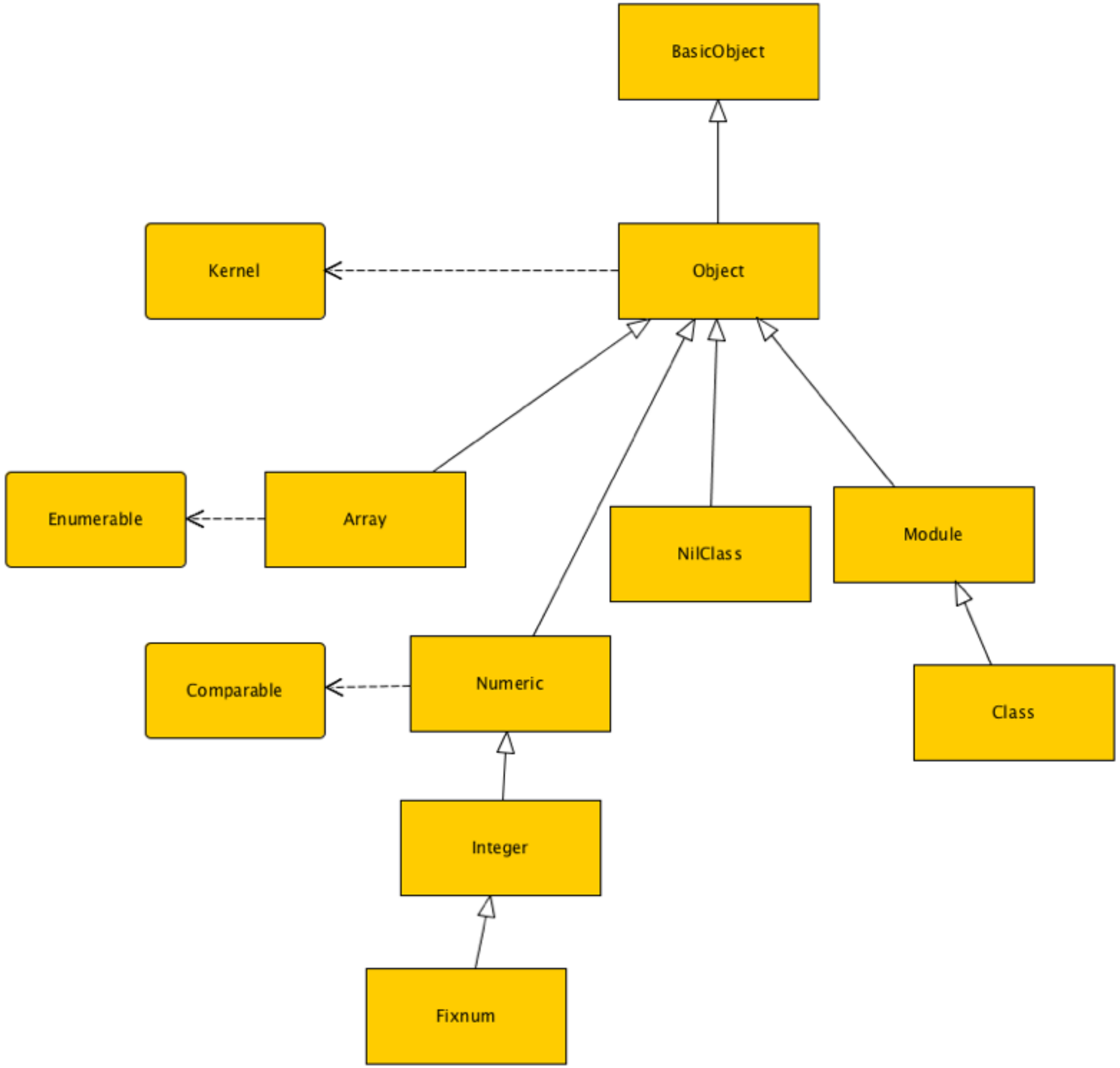
Arma una jerarquía de clases con:

[ ]

3

nil

Hash # objeto tipo Class


BasicObject
Kernel
Object
Enumerable
Array
NilClass
Module
Comparable
Numeric
Class
Integer
Fixnum

MÓDULOS Y CLASES

```ruby
module Splitable

  def split(using)
    self.to_s.split(using)
  end
end



class Person
  attr_accessor :name, :lastname

  def to_s
    "#{lastname}, #{name}"
  end
end
```

```ruby
module Splitable

  def split(using)
    self.to_s.split(using)
  end
end


class Person

  include Splitable

  attr_accessor :name, :lastname

  def to_s
    "#{lastname}, #{name}"
  end
end
```

Un módulo es como un trozo de clase

No es una clase

No se puede instanciar

Clases -> sustantivos

Módulos -> adjetivos

TAREA 2

IIC2513 - TECNOLOGÍAS Y APLICACIONES WEB

I-2016