



Nicolás Bello Camilletti
@nbellocam

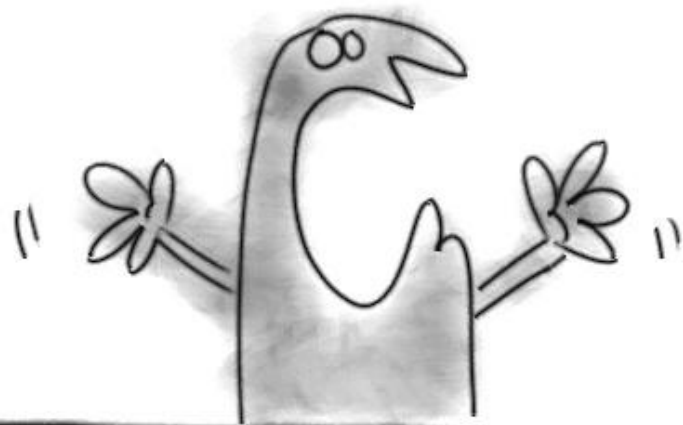
¿Que es Groovy?

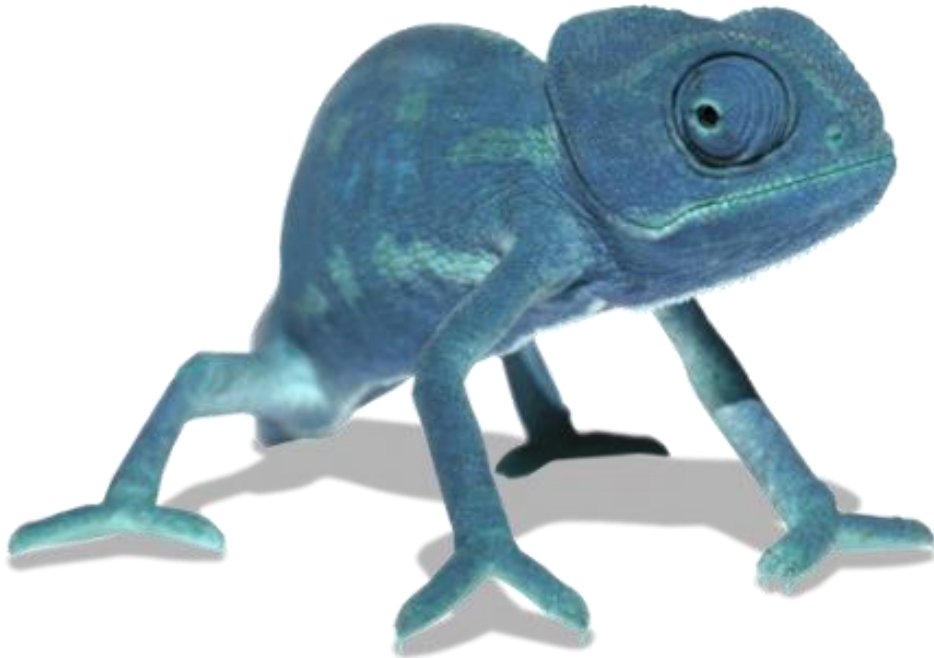




Esto era Groovy

Now What?!!





Lenguaje ágil y
dinámico para la JVM

Integración con Java

Java

Groovy

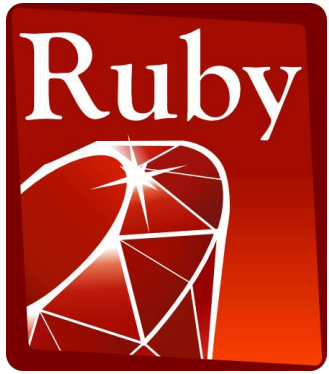
Scala

...

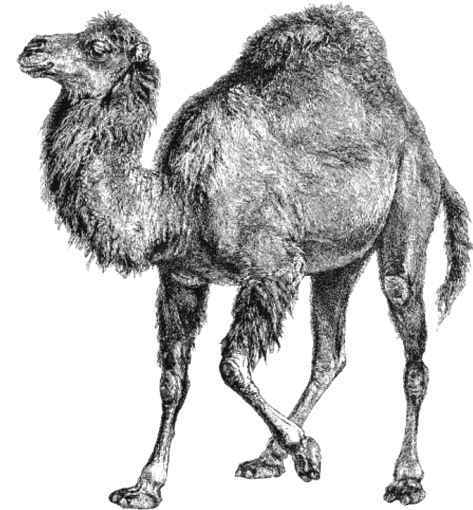
JVM



Curva de aprendizaje
casi nula

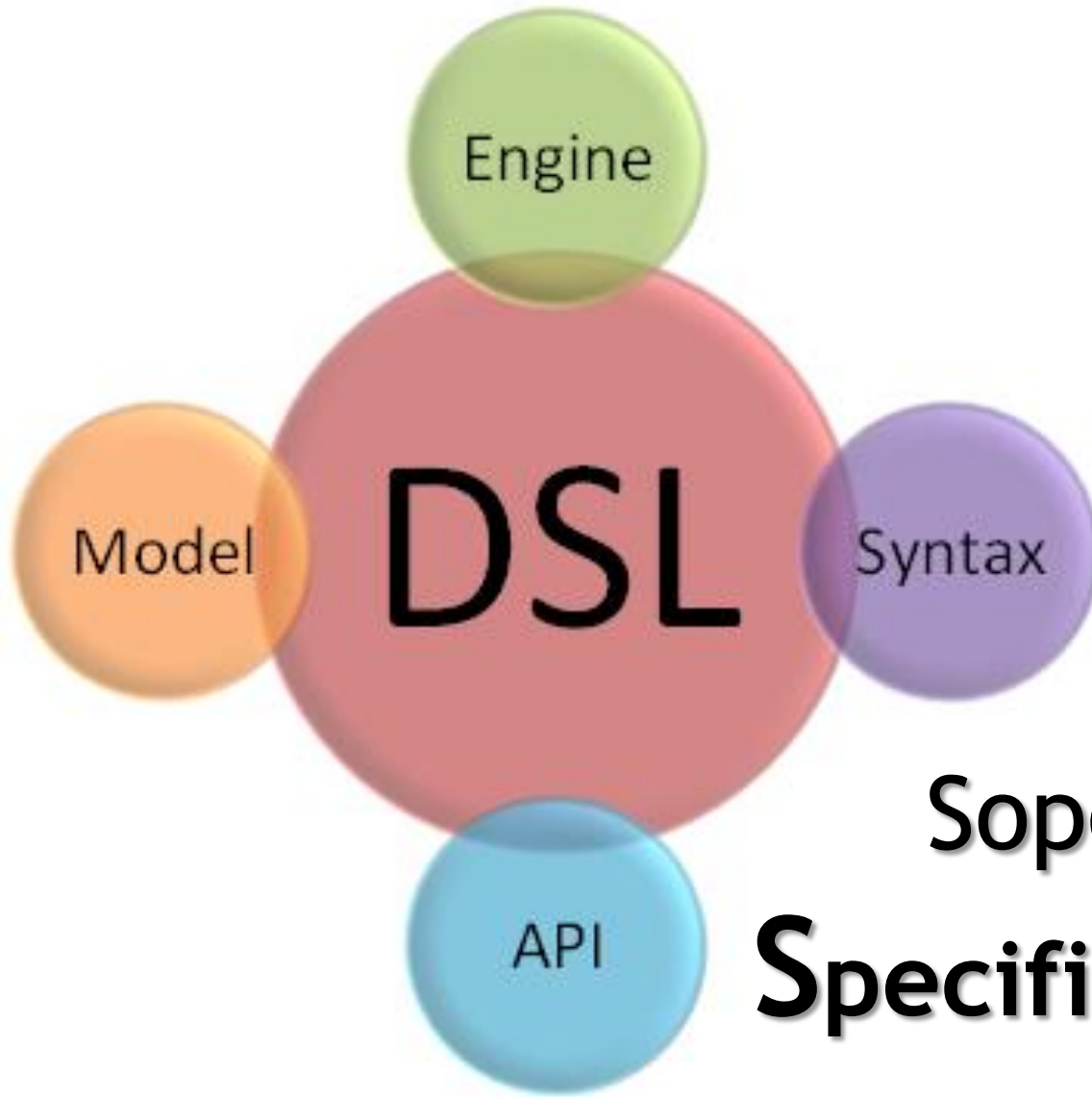


python



Facilita la escritura
de **scripts**



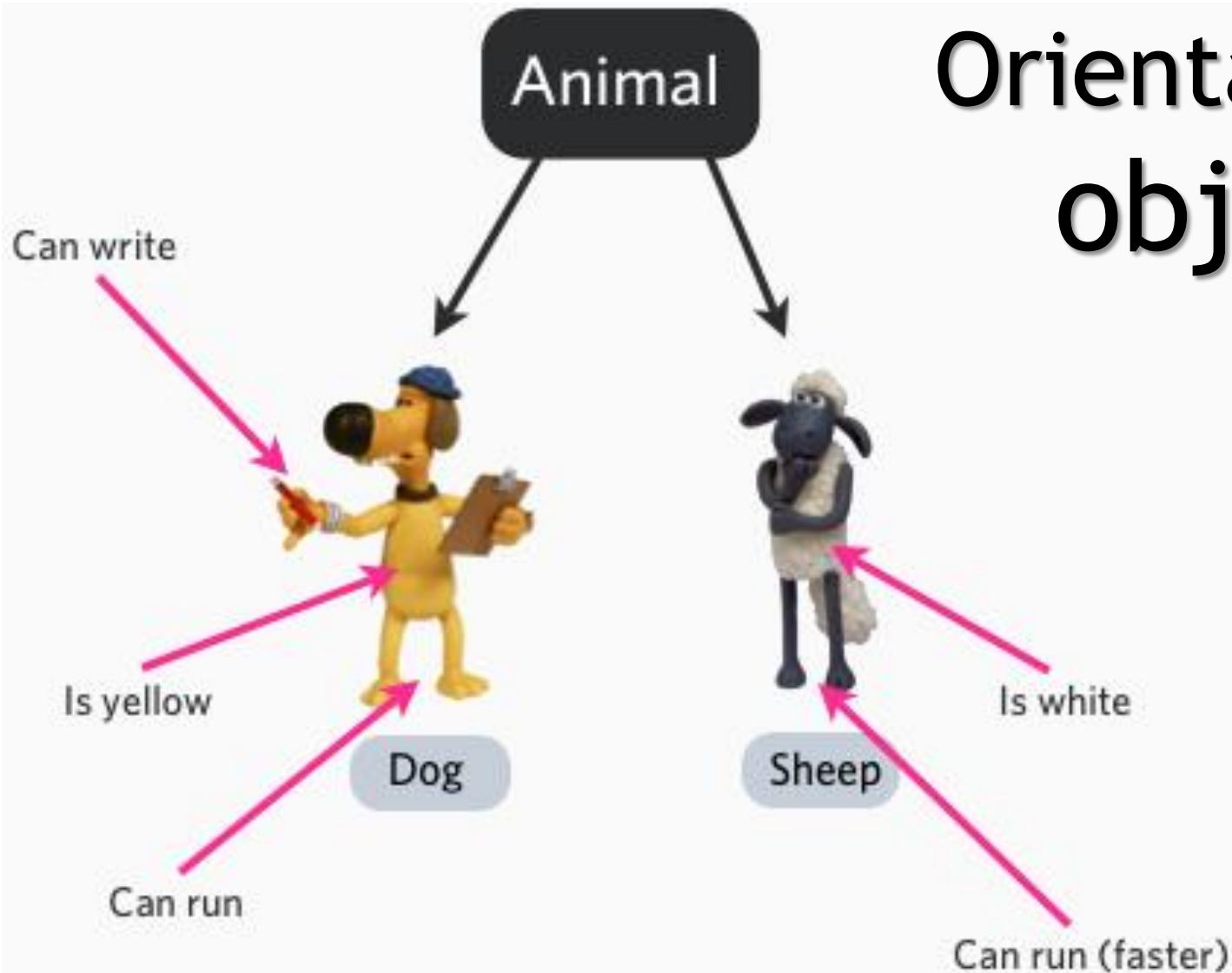


Soporta **Domain**
Specific Languages

Fuertemente tipado



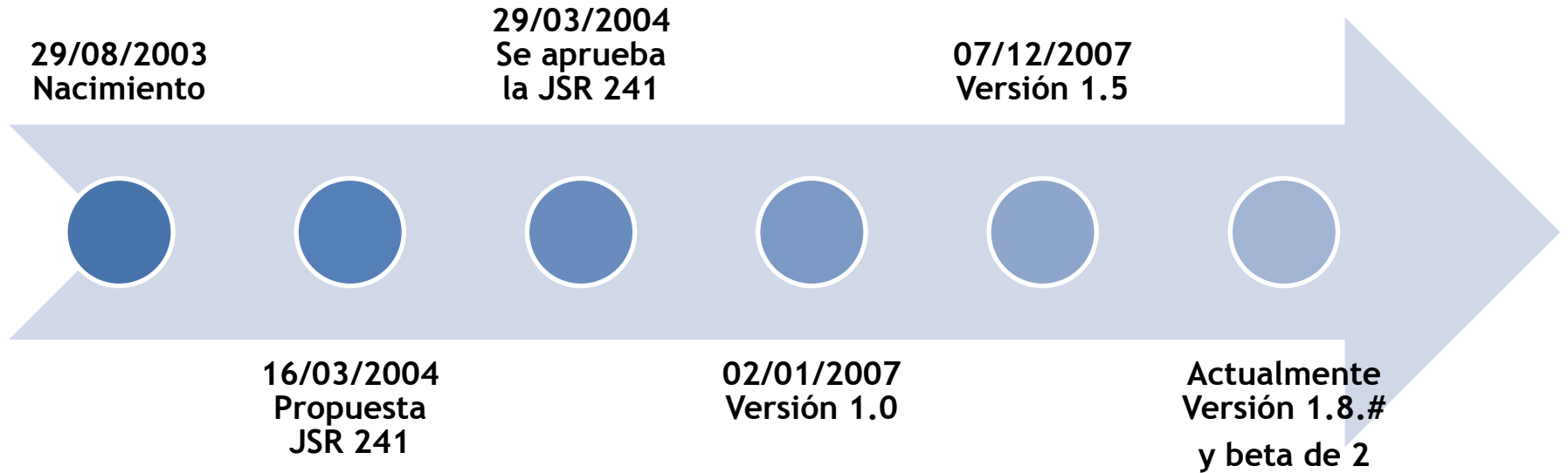
Orientado a objetos




Duck Typing



Historia



A row of colorful LEGO bricks (red and yellow) arranged in a line on a white surface. The bricks are arranged in a slightly curved line, with a yellow brick in the center and red bricks on either side. The background is a plain white surface.

**Algunas
particularidades**

Variables

- Tipado estático
 - Escribiendo el tipo explícitamente

```
String texto  
int contador
```

- Tipado dinámico
 - Utilizando la palabra reservada **def**

```
def variable = "Texto"  
variable = new Customer()  
def method(def param){}
```

Sistema de Tipos

- Numeros Enteros
 - Integer
 - Long
 - BigInteger
 - Byte
 - Short
- Punto flotante
 - BigDecimal
 - Float
 - Double
- Boolean
- Strings
- Colecciones
 - List
 - Map
 - Set
- Rangos
- Expresiones Regulares

Colecciones - Listas

```
def lista = ['casa', 21, 1.69] //lista con cosas diferentes

def numbers = [11, 12, 13, 14] // lista de cuatro elementos

numbers [0]      // 11

numbers [-1]     // 14

numbers [0..2]   // [11, 12, 13]

numbers.get(2)   //13

def emptyList = [] //emptyList.size() == 0

emptyList.add(5) //emptyList.size() == 1
```

Colecciones - Maps

```
def map = [name:"Gromit", likes:"cheese", id:1234]

map.get("name")      // "Gromit"

map["name"]          // "Gromit"

map.name              // "Gromit"

def emptyMap = [:]    // emptyMap.size() == 0

emptyMap.put("foo", 5) // emptyMap.size() == 1

emptyMap.other = [23,52] // emptyMap.size() == 2
```

Colecciones - Rangos

```
1900..1999      // siglo XX (rango inclusivo)
2000..  
2100            // siglo XXI(rango exclusivo)
'A'..'D'        //  A, B, C, y D
10..1           //  10, 9, ..., 1
'Z'..'X'        //  Z, Y, y X

def range = 1..10

assert range.from == 1

assert range.to == 10
```

Strings y GStrings

```
def age = 25

'My age is ${age}'           // My age is ${age}

"My age is ${age}"          // My age is 25

"""My age
  is ${age}"""              /* My age
                             is 25*/
```

Expresiones Regulares

```
//Creación de una expresión regular
def cheese = ~/cheese/

def nicecheese = ("cheesecheese" =~ /cheese/).replaceFirst("nice")
// nicecheese

if("cheese" =~ /cheese/){
    //entra
}else{
    //no entra
}
```


Groovy Truth

- Boolean
- Cosas que son false
 - Colecciones vacías(listas, maps)
 - Iteradores y enumeradores sin mas elementos
 - Strings vacios
 - Matchers de regex que no coinciden (=~)
 - Cero
 - null

Métodos

```
def greetings(def salutation, def name = 'Ken') {  
    println "${salutation} ${name}"  
    return "ok"  
}  
  
greetings('Hello', 'John')           // Hello John  
println greetings('Welcome')          /* Welcome Ken  
                                       ok*/  
  
String estatica(String name){  
    "hello" + name                    // Hello ${name}  
}
```

Closures (1)

- { [param1, param2 ...] -> sentencia1; sentencia2... }

```
def clos = { p -> println "Hello ${p}" }    //con parámetros
def clos2 = { println "Hello ${it}" }        //parámetro actual

clos.call('world')    // el argumento actual es 'world'
clos('shortcut')      // forma abreviada
```

Closures (2)

```
def greeting = 'Hello'

def closLikeLambda = {param -> println "${greeting} ${param}"}

closLikeLambda.call('world')           //Hello world

greeting = 'Welcome'

closLikeLambda.call('world')           // Welcome world
```

Ciclos con closures (1)

- Each

```
stringList.each() { print "${it}" }; println "";  
stringMap.each() { key, value -> println "${key} => ${value}" };
```

- EachWithIndex

```
stringList.eachWithIndex() { obj, i -> println " ${i}: ${obj}" };
```

- Times

```
def len = 10  
len.times { println it;}
```

Ciclos con closures (2)

- Collect

```
def words = ['ant', 'buffalo', 'cat', 'dinosaur']  
assert words.collect{ it[0] } == ['a', 'b', 'c', 'd']
```

- FindAll

```
assert words.findAll{ w -> w.size() > 4 } == ['buffalo', 'dinosaur']
```

Operadores (1)

- Colección*.operación → Spread Operator

```
assert [1, 3, 5] == ['a', 'few', 'words']*.size()
```

- objeto?.operación → Safe Navigation Operator

```
objeto?.operation()
```

– Similar a: `(objeto != null) ? objeto.operation() : null`

Operadores (2)

- objeto?:valorDefault → Elvis Operator

```
def displayName = user.name ?: "Anonymous"
```

– Operador ternario equivalente

```
def displayName = user.name ? user.name : "Anonymous"
```

Orientación a Objetos

```
class Cuenta {  
    def numero // numero de cuenta  
    def balance // balance actual  
  
    def Cuenta(monto, numero) {  
        balance += monto  
        this.numero = numero  
    }  
    def debitar(monto) { // Solo si hay suficiente monto  
        if(balance >= monto) balance -= monto  
    }  
}  
  
def cuenta = new Cuenta(30, 'ABC255')
```

GroovyBeans (1)

```
class Person {  
  
    // properties  
    Integer id  
  
    String name  
  
    Date dob  
  
}  
  
def person = new Person(id:1, name:"Gromit", dob:new Date())  
  
println("Hello ${person.name}")
```

GroovyBeans (2)

- Con access modifier
 - Field
- Sin access modifier
 - Field privado
 - Mas getter y setter públicos (una property)
 - Se pueden sobrescribir
- Property final
 - Field privado como final
 - Solo getter

Expando

```
def player = new Expando()  
  
player.name = "Dierk"  
  
player.greeting = { "Hello, my name is $name" }  
  
println player.greeting()  
  
player.name = "Jochen"  
  
println player.greeting()
```

Implementando Interfaces con Map

```
impl = [  
    i: 10,  
    hasNext: { impl.i > 0 },  
    next: { impl.i-- },  
]  
  
iter = impl as Iterator  
  
while ( iter.hasNext() )  
    println iter.next()
```

Domain Specific Languages

- Muy buen soporte
- Concepto de Builders
- Sobrecarga de operadores
- Closures como parámetros
- Y muchos mas...

Soporte para Lenguajes de Marcado

```
def builder = new groovy.xml.MarkupBuilder()

builder.<u>stocks</u> {

    <u>stock</u>(symbol: 'JAVA')

    <u>stock</u>(symbol: 'MSFT')

    <u>stock</u>(symbol: 'IBM' )

}
```

```
<stocks>

  <stock symbol='JAVA' />

  <stock symbol='MSFT' />

  <stock symbol='IBM' />

</stocks>
```

Diferencias con java

- Defaults imports
- == es equals
- Return opcional
- This en métodos estáticos para clase
- The Groovy way... 😊

Ejemplos

- Java

```
String[] list = new String[] {"Rod", "Carlos", "Chris"};

for (String item : list) {
    if (item.length() <= 4) System.out.println(item);
}
```

- Groovy

```
["Rod", "Carlos", "Chris"].findAll{it.size() <= 4}.each{println it}
```

Ejemplo concurrencia

- Groovy

```
new Thread(  
    {println "running"} as Runnable  
).start()
```

- Java

```
class Hilo extends Thread{  
    def Run() { println "running"}  
}  
  
def unHilo = new Hilo()  
  
unHilo.start()
```

Programación de GUI

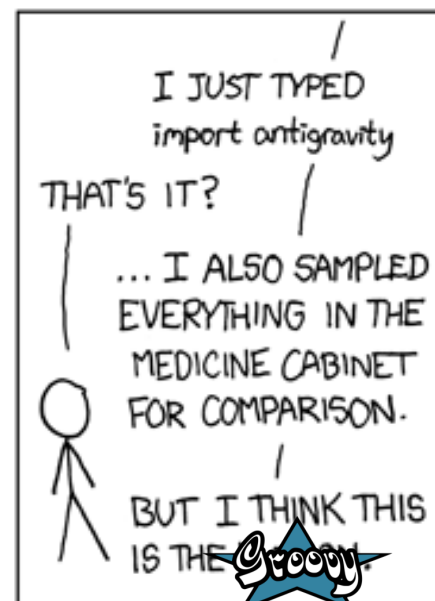
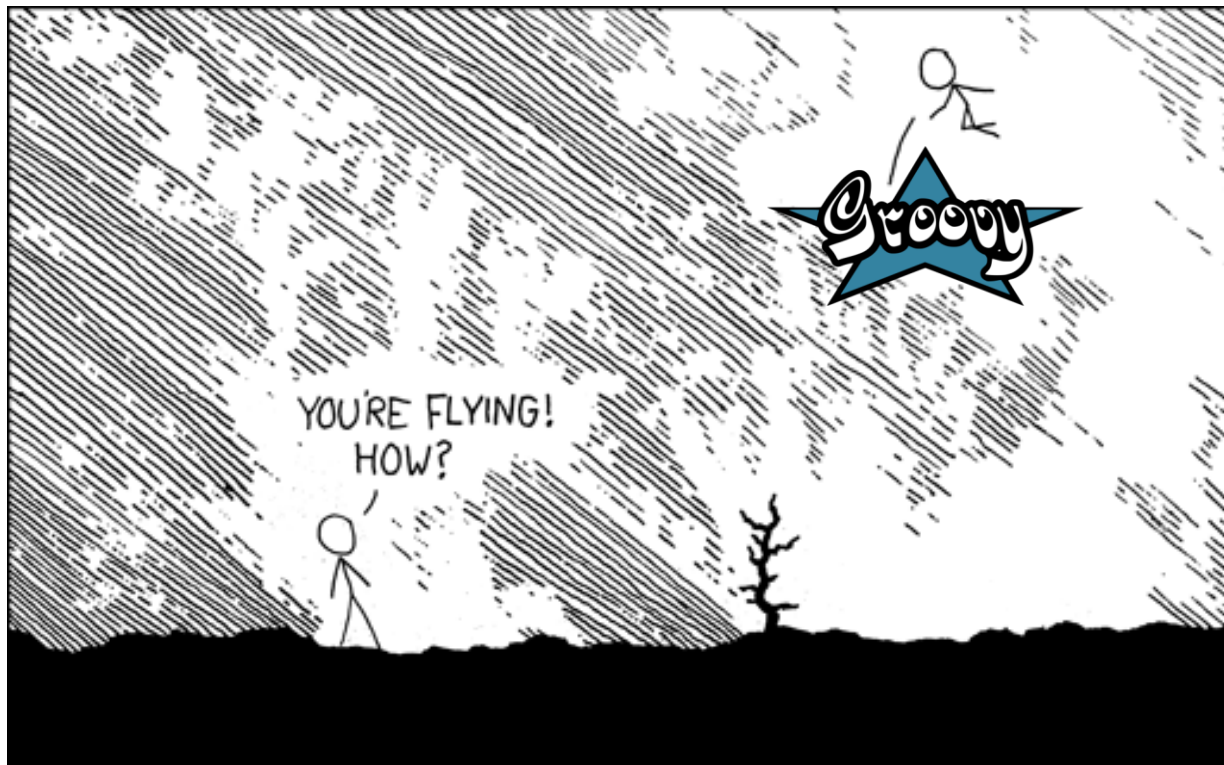
- Wrapper de Swing → Usa Builders

```
count = 0
new groovy.swing.SwingBuilder().edt {
    frame(title:'Frame', size:[300,300], show: true) {
        BorderLayout()
        textlabel = label(text:"Click the button!",
            constraints: java.awt.BorderLayout.NORTH)
        button(text:'Click Me',
            actionPerformed: {
                count++
                textlabel.text = "Clicked ${count} time(s)."
                println "clicked"},
            constraints:java.awt.BorderLayout.SOUTH)
    }
}
```



Conclusiones






Usar Groovy en:

- Integración de componentes
- Modelos de negocio muy dinámicos
- Pruebas de concepto
- Proyectos rápidos y chicos
- Scripting

No usar Groovy en:

- Algoritmos complejos
 - Cálculos intensivos
- Manejo de grandes cantidades de datos
 - Problema con creación de objetos por ser dinámico
- Sistemas con
 - Requerimientos bien definidos
 - Pocas posibilidades de evolución
- Proyectos muy grandes

An illustration on a reddish-brown textured background. Three men are pushing large, light-colored rectangular blocks. One man in a purple suit is pushing a block towards the left. Another man in a light green shirt and blue pants is pushing a block towards the right. A third man in a dark suit is rolling a large, textured sphere towards the right. In the lower-left area, there is a pile of small, dark, circular objects, possibly coins or stones, with a small stick or stick-like object lying next to them.

**No trabajan
duro.**

**Trabajen de forma
inteligente!**

Recursos

- Sitio oficial
 - <http://groovy.codehaus.org/>



Pruébenlo!



Muchas Gracias!!!