A Rubyist's Guide to Smalltalk

https://github.com/ngty/levelo.git

~ ngtzeyang

hplsg

Smalltalk's history

One fine day in the early 1970s in Xerox PARC ~
Alan Kay bullishly **betted** that the
"most powerful language in the world" can be implemented
"in a page of code" ...

Why smalltalk ?

Curiosity, pragmatic programmers say that we should learn a language every year, smalltalk is ahead of its time, Kent Beck: **"I always knew that one day smalltalk would replace java, I just didn't know it would be called Ruby"**, smalltalk has many influences ~ OOP, IDE, SUnit, WIMP interfaces, etc ...

Basics

Keywords

Smalltalk: #6

```
true, false, nil,
self, super,
thisContext
```

Ruby: #41

```
class, module, def, undef,
begin, rescue, ensure, end, if,
unless, then, elsif, else,
case, when, while, until, for,
break, next, redo, retry,  in,
do, return, yield, super, self,
nil, true, false, and, or, not,
alias, defined, BEGIN, END,
__LINE__, __FILE__,
__ENCODING__
```

cont.) Keywords (re-expressing 'if')

Smalltalk:

```
01 (2 > 1)
02    ifTrue:  ['BIGGER']
03    ifFalse: ['otherwise'].
 >> 'BIGGER'
```

Ruby:

```
01 if 2 > 1
02   'BIGGER'
03 else
04   'otherwise'
05 end
 >> 'BIGGER'
```

Everything is an object

Smalltalk ~ everything, including keywords

```
01 2011 class.
  >> SmallInteger

02 #smalltalk class.
  >> Symbol
```

***Ruby* ~ everything, except for most keywords**

```
01 2011.class
  >> Fixnum

02 :ruby.class
  >> Symbol
```

Its about message passing ...

"You never tell an object what to do ... instead you politely ask it to do something by sending it a message. The object, not you, selects the appropriate method for responding to your message."

~ Squeak by Example

When the message makes sense

Smalltalk:

```
01 'thing' asUppercase.
 >> 'THING'

02 'thing' perform: ('as',
03   'Uppercase') asSymbol.
 >> 'THING'
```

Ruby:

```
01 'thing'.upcase
 >> 'THING'

02 'thing'.send \
03   ('up' + 'case')
 >> 'THING'
```

When the message makes no sense

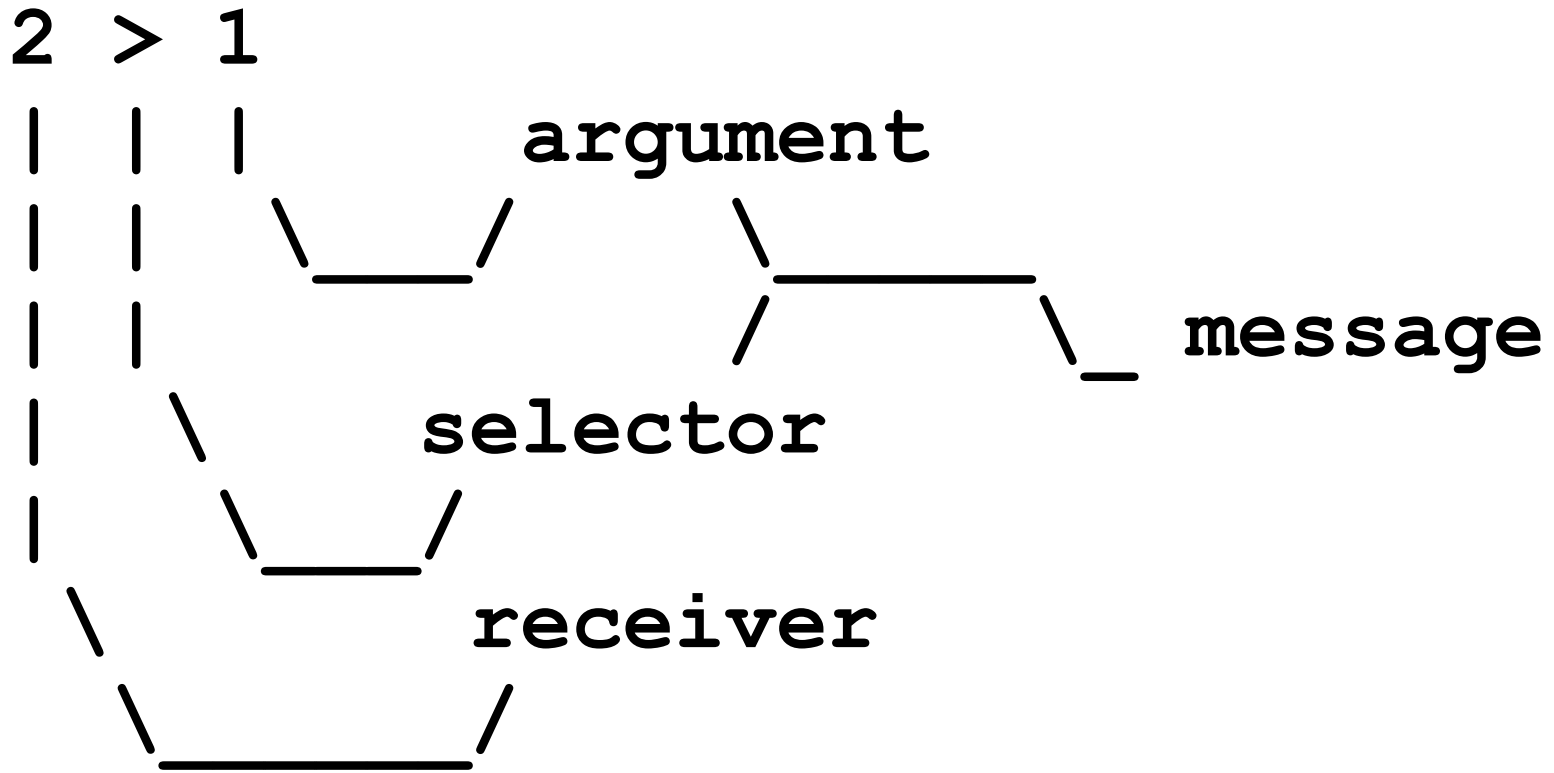
Smalltalk ~ eventually handled by #doesNotUnderstand, & ...

```
01 2001 asUppercase.
 >> MessageNotUnderstood
```

Ruby **~ eventually handled by #method_missing, & ...**

```
01 2001.upcase
 >> NoMethodError
```

Breaking down message passing ...



Message Type 1/3: Unary

- takes no argument
- selectors are #subStrings & #isArray

```
01 'red dot ruby conf' subStrings.
 >> #('red' 'dot' 'ruby' 'conf')

02 'red dot ruby conf' isString.
 >> true
```

Message Type 2/3: Binary

- maximum of 2 non-alphanumeric characters
- takes an argument
- selectors are #<= & #, respectively

```
01 1 <= 3
 >> true

02 'reddot', 'rubyconf'.
 >> 'reddotrubyconf'
```

Message Type 3/3: Keyword(s)

- each keyword has an appending ':'
- each keyword takes an argument
- one or more keywords
- selectors are #collect: & #ifTrue:ifFalse: respectively

```
01 #(1 2) collect: [:e| e + 1 ].
 >> #(2 3)

02 (2 > 1) ifTrue: [ 'BIGGER' ]
03   ifFalse: [ 'otherwise' ].
 >> Thing
```

Chaining Messages

- return of one message becomes receiver for the next message
- precedence rule applies: unary, then binary, & keyword(s)

```
01 2 squared + 4 squared between: 1 and: 20.
 >> true

02 ((2 squared) +
03    (4 squared)) between: 1 and: 20.
 >> true
```

cont.) Chaining Messages

- use parenthesis to change precedence

```
01 (2 squared + 4) squared
02   between: 1 and: 20.
 >> false

03 ((2 squared + 4) squared)
04   between: 1 and: 20.
 >> false
```

Cascading Messages

- use semi-colon (;) to separate messages
- only receiver in the 1st message specified
- subsequent messages assume the same receiver

```
01 SortedCollection new
02    add: #red;
03    add: #blue;
04    yourself.
 >> SortedCollection(#blue #red)
```

Class Matters

Self is never implicit !!

Smalltalk:

```
01 Object subclass: Thing [
02    whoami [
03       ^ self description
04    ]
05    description [
06       ^ 'i''m thingy'
07    ]
08 ]
09
10 Thing new whoami.
 >> 'i''m thingy'
```

Ruby:

```
01 class Thing
02  def whoami
03   description
04  end
05  def description
06   "i'm thingy"
07  end
08 end
09
10 Thing.new.whoami
 >> "i'm thingy"
```

What or who is super ??

Smalltalk:

```
01 Object subclass: Thing [
02   whoami [ ^ 'thing' ]
03 ]
04 Thing subclass: Tree [
05   whoami [
06     ^ super whoami,
07       ' / tree'
08   ]
09 ]
10
11 Tree new whoami.
 >> 'thing / tree'
```

Ruby:

```
01 class Thing
02  def whoami
03   'thing'
04  end
05 end
06 class Tree < Thing
07  def whoami
08   "#{super} / tree"
09  end
10 end
11
12 Tree.new.whoami
 >> "thing / tree"
```

Open Class

Smalltalk:

```
01 Object subclass: #Thing.
02 thing := Thing new.
03 thing whoami.
 >> MessageNotUnderstood

04 Thing extend [
05    whoami [ ]
06 ]
07 thing whoami
 >> a Thing
```

Ruby:

```
01 class Thing; end
02 thing = Thing.new
03 thing.whoami
 >> NoMethodError

04 class Thing
05  def whoami
06   self
07  end
06 end
07 thing.whoami
 >> <#Thing:???>
```

Metaclass

```
Metaclass class
     \
      |> Object subclass: #T.
     /
T class (a Metaclass instance) &
 T (a T class singleton instance)
     \
      |> t := T new.
     /
t (a T instance)
```

cont.) Metaclass (new class)

```
Metaclass class
     \
      |> Object subclass: #T.
     /
T class (a Metaclass instance) &
 T (a T class singleton instance)
     \
      |> t := T new.
     /
t (a T instance)
```

cont.) Metaclass (new instance)

```
Metaclass class
     \
      |> Object subclass: #T.
     /
T class (a Metaclass instance) &
 T (a T class singleton instance)
     \
      |> t := T new.
     /
t (a T instance)
```

Metaprogramming

Smalltalk:

```
01 Object subclass: #T.
02 meth := 'say [ ^ #hi ]'.
03 T compile: meth.
04 T new say.
 >> #hi
```

Ruby:

```
01 T  = Class.new
02 T.class_eval do
03   def say; :hi; end
04 end
05 T.new.say
 >> :hi
```

cont.) Metaprogramming

Smalltalk:

```
01 Object subclass: #T.
02 meth := 'say [ ^ #HI ]'.
03 T class compile: meth.
04 T say.
 >> #HI
```

Ruby:

```
01 T  = Class.new
02 T.instance_eval do
03   def say; :HI; end
04 end
05 T.say
 >> :HI
```

Summing Up

Smalltalk vs Ruby

Smalltalk >

```
#('simple' 'pure') join: ' & '
```

Ruby >

```
%w{pragmatic fun} * ' & '
```

Interesting resources

- Environments:
  - pharo @ http://www.pharo-project.org/home
  - squeak @ http://www.squeak.org/
  - gnu smalltalk @ http://smalltalk.gnu.org/
- Web application frameworks:
  - seaside @ http://www.seaside.st/
  - aida/web @ http://www.aidaweb.si/
- Meta-data description frameworks:
  - magritte @ http://www.lukas-renggli.ch/smalltalk/magritte
- Ruby on smalltalk vm:
  - maglev @ http://ruby.gemstone.com/