INRIA

RMod

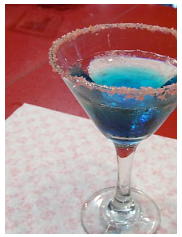
A little journey in a dynamic world

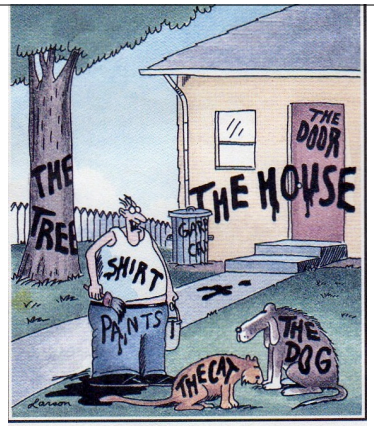
Stéphane Ducasse
stephane.ducasse@inria.fr
http://stephane.ducasse.free.fr/

Goal

Lower your stress :)
Show you that this is simple

Appetizer!





"Now! ... *That* should clear up a few things around here!"

Yeah!

Smalltalk is a dynamically typed language


```
ArrayList<String> strings
        = new ArrayList<String>();
```

strings := ArrayList new.

Shorter

```
Thread regThread = new Thread(
  new Runnable() {
    public void run() {
        this.doSomething();}
        });
regThread.start();
```

[self doSomething] fork.

Smalltalk = Objects + Messages + (...)

no Math.sin(0.7)
just 0.7 sin

Roadmap

Fun with numbers

Automatic coercion?

1 class

1 class

>SmallInteger

1 class maxVal

1 class maxVal

>1073741823

1 class maxVal + 1

1 class maxVal + 1

>1073741824

(1 class maxVal + 1) class

(1 class maxVal + 1) class

>LargePositiveInteger

(1/3) + (2/3)

(1/3) + (2/3)

>1

2/3 + 1

2/3 + 1

> 5/3

1000 factorial

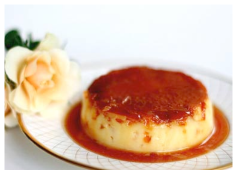
1000 factorial

4023872600770937735437024339230039857193748642107146325437999104299385123986290205920442084869694048004799
8861019719605863166687299480855890132382966994459099742450408707375991882362772718873251977950595099527612
0874975462497043601418278094646496291056393887437886487337119181045825783647849977012476632889835955735432
5131853239584630755574091142624174743493475534286465766116677973966688202912073791438537195882498081268678
3837455973174613608537953452422158659320192809087829730843139284440328123155861103697680135730421616874760
9675871348312025478589320767169132448426236131412508780208000261683151027341827977704784635868170164365024
1536913982812648102130927612448963599287051149649754199093422215668325720808213331861168115536158365469840
4670897560290095053761647584772842188967964624494516076535340819890138544248798495995331910172335555660213
9450399736280750137837615307127761926849034352625200015888535147331611702103968175921510907788019393178114
1945452572238655414610628921879602238389714760885062768629671466746975629112340824392081601537808898939645
1826324367161676217916890977991190375403127462228998800519544441428201218736174599264295658174662830295557
0299024324153181617210465832036786906117260158783520751516284225540265170483304226143974286933061690897968
4825901254583271682264580665267699586526822728070757813918581788896522081643483448259932660433676601769996
1283186078838615027946595513115655203609398818061213855860030143569452722420634463179746059468257310379008
4024432438465657245014402821885252470935190620929023136493273497565513958720559654228749774011413346962715
4228458623773875382304838656889764619273838149001407673104466402598994902222217659043399018860185665264850
6179970235619389701786004081188972991831102117122984590164192106888438712185564612496079872290851929681937
2388642614839657382291123125024186649353143970137428531926649875337218940694281434118520158014123344828015
0513996942901534830776445690990731524332782882698646027898643211390835062170950025973898635542771967428222
4875758676575234422020757363056949882508796892816275384886339690995982628095612145099487170124451646126037
9029309120889086942028510640182154399457156805941872748998094254742173582401063677404595741785160829230135
3580818400969963725242305608559037006242712434169090041536901059339838357779394109700277534720000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000

1000 factorial / 999 factorial

1000 factorial / 999 factorial

> 1000



Roadmap

A first program

$C $h $a $r $a $c $t $e $r

$F, $Q $U $E $N $T $i $N

space tab?!

Character space
Character tab
Character cr

‘Strings’

‘Tiramisu’

Characters

12 printString

> ’12’

Strings are collections of chars

‘Tiramisu’ at: 1

Strings are collections of chars

‘Tiramisu’ at: 1

> $T

A program! -- finding the last char

A program!

| str |

A program!

| str |

local variable

A program!

```
| str |
str := 'Tiramisu'.
```

local variable

A program!

```
| str |
str := 'Tiramisu'.
```

local variable
assignment

A program!

```
| str |
str := 'Tiramisu'.
str at: str length
```

local variable
assignment

A program!

```
| str |
str := 'Tiramisu'.
str at: str length
```

local variable
assignment
message send

```
> $u
```

Syntax Summary

comment:	"a comment"
character:	$c $h $a $r $a $c $t $e $r $s $# $@
string:	'a nice string' 'lulu' 'l''idiot'
symbol:	#mac #+
array:	#(1 2 3 (1 3) $a 4)
byte array:	#[1 2 3]
integer:	1, 2r101
real:	1.5, 6.03e-34,4, 2.4e7
fraction:	1/33
boolean:	true, false
point:	10@120

Roadmap

Fun with keywords-based messages

Keyword-based messages

arr **at:** 2 **put:** 'loves'

Keyword-based messages

arr **at:** 2 **put:** 'loves'

somehow like arr.atput(2,'loves')

From Java to Smalltalk

```
postman.send(mail,recipient);
```

Removing

postman.send(mail,recipient);

Removing unnecessary

postman send mail recipient

But without losing information

postman send mail to recipient

postman send: mail to: recipient

postman.send(mail,recipient);

postman send: mail to: recipient

postman.send(mail,recipient);

The message is send:to:

Roadmap

Fun with classes

A class definition!

Superclass subclass: #**Class**
instanceVariableNames: ***'a b c'***
...
category: 'Package name'

A class definition!

Object* subclass: #*Point
instanceVariableNames: *'x y'*
classVariableNames: ''
poolDictionaries: ''
category: *'Graphics-Primitives'*

A class definition!

Object* subclass: #*Point
instanceVariableNames: *'x y'*
classVariableNames: ''
poolDictionaries: ''
category: *'Graphics-Primitives'*

Roadmap

Fun with methods

On Integer

```
factorial
	"Answer the factorial of the receiver."

	self = 0 ifTrue: [^ 1].
	self > 0 ifTrue: [^ self * (self - 1) factorial].
	self error: 'Not valid for negative integers'
```


Summary

self, super
can access instance variables
can define local variable | ... |
Do not need to define argument types
^ to return

Roadmap

Fun with unary messages

1 class

1 class

> SmallInteger

false not

false not

> true

Date today

Date today
> 24 May 2009

Time now

Time now
> 6:50:13 pm

Float pi

Float pi
> 3.141592653589793

We sent messages to objects or classes!

1 class
Date today

We sent messages to objects or classes!

1 class
Date today

Roadmap

Fun with binary messages

aReceiver aSelector anArgument

Used for arithmetic, comparison and logical operations
One or two characters taken from:
+ - / \ * ~ < > = @ % | & ! ? ,

1 + 2

1 + 2
>3

2 => 3

2 => 3
> false

10 @ 200

'Black chocolate' ,' is good'

Roadmap

Fun with keyword-based messages

10@20 setX: 2

10@20 setX: 2
> 2@20

12 between: 10 and: 20

12 between: 10 and: 20
> true

receiver
keyword1: argument1
keyword2: argument2

equivalent to
receiver.keyword1keyword2(argument1, argument2)

receiver
keyword1: argument1
keyword2: argument2

equivalent to
receiver.keyword1keyword2(argument1, argument2)

Roadmap

Messages messages
messages
again messages
....

Yes there are only messages
unary
binary
keywords

Composition: from left to right!

69 class inspect

69 class superclass superclass inspect

Precedence

Unary> Binary> Keywords

2 + 3 squared

2 + 3 squared
> 2 + 9

2 + 3 squared
> 2 + 9
> 11

Color gray - Color white = Color black

Color gray - Color white = Color black
> aColor = Color black

Color gray - Color white = Color black
> aColor = Color black
> true

2 raisedTo: 3 + 2

2 raisedTo: 3 + 2
> 2 raisedTo: 5

2 raisedTo: 3 + 2
> 2 raisedTo: 5
> 32

No mathematical precedence

1/3 + 2/3

No mathematical precedence

1/3 + 2/3
>7/3 /3

(Msg) > Unary> Binary> Keywords

Parenthesized takes precedence!

(0@0 extent: 100@100) bottomRight

(0@0 extent: 100@100) bottomRight
> (aPoint extent: anotherPoint)
bottomRight

(0@0 extent: 100@100) bottomRight
> (aPoint extent: anotherPoint)
bottomRight
> aRectangle bottomRight

(0@0 extent: 100@100) bottomRight
> (aPoint extent: anotherPoint)
bottomRight
> aRectangle bottomRight
> 100@100

0@0 extent: 100@100 bottomRight

0@0 extent: 100@100 bottomRight
> Message not understood
> 100 does not understand bottomRight

Only Messages

(Msg) > Unary > Binary > Keywords
from left to right
No mathematical precedence

Roadmap

Fun with blocks

Function definition

fct(x) = x * x + x

Function Application

fct (2) = 6
fct (20) = 420

Function definition

fct(x) = x * x + x

|fct|
fct:= **[:x** | x * x + x**]**.

Function Application

fct (2) = 6
fct (20) = 420

fct value: 2
> 6
fct value: 20
> 420

Other examples

[2 + 3 + 4 + 5] value
[:x | x + 3 + 4 + 5] value: 2
[:x :y | x + y + 4 + 5] value: 2 value: 3

Block

anonymous method

[*:variable1 :variable2* **|**
| *tmp* |
expression1.
...variable1 ... **]**

value: ...

Block

anonymous method
Really really cool!
Can be passed to methods, stored in instance variables

[*:variable1 :variable2* **|**
| *tmp* |
expression1.
...variable1 ... **]**

value: ...

Roadmap

Fun with conditional

Example

```
3 > 0
  ifTrue:['positive']
  ifFalse:['negative']
```


Example

```
3 > 0
  ifTrue:['positive']
  ifFalse:['negative']
```

> 'positive'

Yes ifTrue:ifFalse: is a message!

```
Weather isRaining
    ifTrue: [self takeMyUmbrella]
    ifFalse: [self takeMySunglasses]
```

ifTrue:ifFalse is sent to an object: a boolean!

Booleans

& | not
or: and: (lazy)
xor:
ifTrue:ifFalse:
ifFalse:ifTrue:
...

Yes! ifTrue:ifFalse: is a message send to a Boolean.

But optimized by the compiler :)

Conditions are messages sent to boolean

```
(x isBlue) ifTrue: [  ]
```

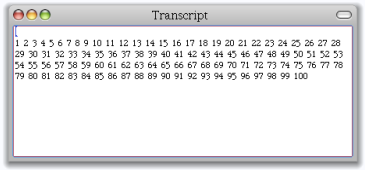
Roadmap

Fun with loops


```
1 to: 100 do:
    [ :i | Transcript show: i ; space]
```

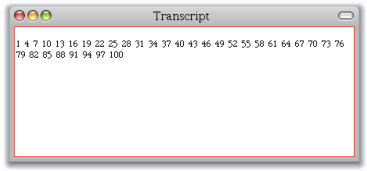


```
1 to: 100 do:
    [ :i | Transcript show: i ; space]
```



1 **to:** 100 **by:** 3 **do:**
[:i | Transcript show: i ; space]

1 **to:** 100 **by:** 3 **do:**
[:i | Transcript show: i ; space]



So yes there are real loops in Smalltalk!

to:do:
to:by:do:
are just messages send to integers

So yes there are real loops in Smalltalk!

to:do:
to:by:do:
are just messages send to integers

Roadmap

Fun with iterators


```
ArrayList<String> strings
            = new ArrayList<String>();
for(Person person: persons)
      strings.add(person.name());
```

```
strings :=
persons collect [:person | person name].
```


#(2 -3 4 -35 4) **collect:** [:each| each abs]

#(2 -3 4 -35 4) **collect:** [:each| each abs]
> #(2 3 4 35 4)

#(15 10 19 68) **collect:** [:i | i odd]

#(15 10 19 68) **collect:** [:i | i odd]
> #(true false true false)

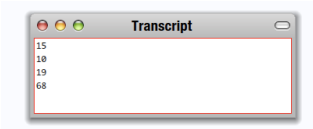
#(15 10 19 68) **collect:** [:i | i odd]

We can also do it that way!

```
|result|
aCol :=  #( 2 -3 4 -35 4).
result := aCol species new: aCol size.
1 to: aCollection size do:
        [ :each | result at: each put: (aCol at: each) odd].
result
```

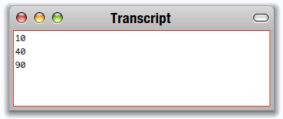

#(15 10 19 68) **do:**
[:i | Transcript show: i ; cr]

#(15 10 19 68) **do:**
[:i | Transcript show: i ; cr]



#(1 2 3)
with: #(10 20 30)
do: [:x :y| Transcript show: (y ** x) ; cr]

#(1 2 3)
with: #(10 20 30)
do: [:x :y| Transcript show: (y ** x) ; cr]



How do: is implemented?

How do: is implemented?

```
SequenceableCollection>>do: aBlock
  "Evaluate aBlock with each of the receiver's elements as the
  argument."

  1 to: self size do: [:i | aBlock value: (self at: i)]
```


Some others... friends

#(15 10 19 68) select: [:i|i odd]

#(15 10 19 68) reject: [:i|i odd]

#(12 10 19 68 21) detect: [:i|i odd]

#(12 10 12 68) detect: [:i|i odd] ifNone:[1]

Some others... friends

#(15 10 19 68) select: [:i|i odd]
> #(15 19)

#(15 10 19 68) reject: [:i|i odd]

#(12 10 19 68 21) detect: [:i|i odd]

#(12 10 12 68) detect: [:i|i odd] ifNone:[1]

Some others... friends

#(15 10 19 68) select: [:i|i odd]
> #(15 19)

#(15 10 19 68) reject: [:i|i odd]
> #(10 68)

#(12 10 19 68 21) detect: [:i|i odd]

#(12 10 12 68) detect: [:i|i odd] ifNone:[1]

Some others... friends

#(15 10 19 68) select: [:i|i odd]
> #(15 19)

#(15 10 19 68) reject: [:i|i odd]
> #(10 68)

#(12 10 19 68 21) detect: [:i|i odd]
> 19

#(12 10 12 68) detect: [:i|i odd] ifNone:[1]

Some others... friends

#(15 10 19 68) select: [:i|i odd]
> #(15 19)

#(15 10 19 68) reject: [:i|i odd]
> #(10 68)

#(12 10 19 68 21) detect: [:i|i odd]
> 19

#(12 10 12 68) detect: [:i|i odd] ifNone:[1]
> 1

Iterators are your best friends

compact
nice abstraction
Just messages sent to collections

Iterators are your best friends

compact
nice abstraction
Just messages sent to collections

A simple exercise

How do you define the method that does that?

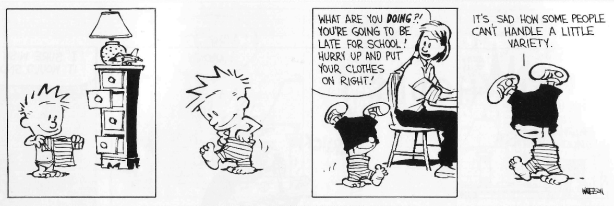
#() -> ''
#(a) ->'a'
#(a b c) -> 'a, b, c'

#(a b c)

do: [:each | Transcript show: each printString]
separatedBy: [Transcript show:',']

#(a b c)

do: [:each | Transcript show: each printString]
separatedBy: [Transcript show:',']



Smalltalk is fun

Pure simple powerful

Check the book
www.pharobyexample.org

www.seaside.st
(www.dabbledb.com)
www.pharo-project.org