A little journey in a dynamic world

Goal
Lower your stress :)  
Show you that this is simple

Appetizer!

Yeah!
Smalltalk is a dynamically typed language

Roadmap
Fun with numbers

Shorter

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

[small doSomething] fork.

Smalltalk = Objects + Messages + (...)  

no Math.sin(0.7)
just 0.7 sin

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

strings := ArrayList new.
Automatic coercion?

I class

I class

> SmallInteger

I class maxVal

I class maxVal

> 1073741823

1 class maxVal + 1

(1 class maxVal + 1) class

> LargePositiveInteger

1 class maxVal + 1

> 1073741824

(1 class maxVal + 1) class

> LargePositiveInteger
(1/3) + (2/3) > 1

2/3 + 1 > 5/3

1000 factorial

1000 factorial / 999 factorial > 1000

1000 factorial / 999 factorial

Roadmap

A first program
Character space
Character tab
Character cr

Strings are collections of chars
'Tiramisu' at: 1
> $T

A program! -- finding the last char
| str |

A program!
| str |

local variable
A program!

| str |
str := 'Tiramisu'.

local variable

Assignment

str at: str length

message send

Syntax Summary

comment:  "a comment"
character:  $c $h $a $r $a $c $t $e $r $s $# $@
string: 'a nice string' 'lulu' 'I'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'

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);

The message is send:to:

postman.send(mail, recipient);

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’
**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
Float pi

> 3.141592653589793

We sent messages to objects or classes!

1 class
Date today

Roadmap

Fun with binary messages

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
**Composition: from left to right!**

- 69 class inspect
- 69 class superclass superclass inspect

**Roadmap**

- Messages messages
- messages again messages

**Yes there are only messages**

- unary
- binary
- keywords

**Precedence**

**Unary** > **Binary** > **Keywords**

- 2 + 3 squared

**12 between: 10 and: 20**

- receiver
  - keyword1: argument1
  - keyword2: argument2
  - equivalent to
    - receiver.keyword1keyword2(argument1, argument2)

- > true
2 + 3 squared
> 2 + 9

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

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

No mathematical precedence

1/3 + 2/3
No mathematical precedence

\[
\frac{1}{3} + \frac{2}{3} > \frac{7}{3} / 3
\]

Parenthesized takes precedence!

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

Message not understood
> 100 does not understand bottomRight

Only Messages

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

fct(x) = x * x + x

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

Function Application

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

Block

anonymous method

[ variable1 | variable2 ]
| tmp | expression1.
...variable1 / ...

value: ...

Other examples

[2 + 3 + 4 + 5] value
[2 + 3 + 4 + 5] value: 2

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

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 sent 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]
So yes there are real loops in Smalltalk!

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

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 ]

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

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) 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]
> 1

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'
Smalltalk is fun

Pure simple powerful

Check the book
www.pharobyexample.org

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