Understanding Messages

Stéphane Ducasse and Damien Cassou

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

Only Objects and Messages

We **only** manipulate objects: mouse, booleans, arrays, numbers, compressed, strings, windows, scrollbars, canvas, files, trees, compilers, sound, url, socket, fonts, text, collections, stack, shortcut, streams, **...**
and we send messages, messages, messages and messages (and closures)

Syntax

- Remember it was originally invented for kids
- Programs look like little sentences
- Minimizing the number of parenthesis

Example

```
(ZnEasy getPng: 'http://a.tile.openstreetmap.org/8/126/87.png' asZnUrl)
   asMorph openInWorld
```

Three Kinds of Messages

- Unary
  - `1 class`, `Browser open`
- Binary (operators like)
  - `1+2`, `x ~~ nil`
- Keyword-based:
  - `2 between: 0 and: 5`

A Glimpse at Message Precedence

- (Msg) $>$ Unary $>$ Binary $>$ Keywords

- First we execute `()`
- Then unary, then binary and finally keyword messages
- Minimize `()` needs
- But let us start with messages

Guess!

- `1 log`
- `Browser open`
- `2 raisedTo: 5`
- `'hello', 'world'`
- `10@20`
- `point1 x`
- `point1 distanceFrom: point2`

Guess

- `1 log` (unary)
- `Browser open` (unary)
- `2 raisedTo: 5` (keyword)
- `’hello’, ’world’` (binary)
- `10@20` (binary)
- `point1 x` (unary)
- `point1 distanceFrom: point2` (keyword)

Unary Messages

```
anObject aSelector
```

```
1 class
> SmallInteger
```

Unary Message Examples

```
false not
> true
```

```
Date today
> 24 May 2009
```

```
Time now
> 6:50:13 pm
```

```
Float pi
> 3.141592653589793
```

Did you notice? To any objects

- We sent messages to any objects!
- We sent messages classes too!
- There is no difference between sending a message to one object or to a class

```
1 class
> SmallInteger
```

```
Date today
> 27 June 2015
```

```
Point selectors
> #(#x #theta #quadrantOf: #onLineFrom:to:within: #bitShiftPoint: #< #scaleFrom:to: #sideOf: #'\\'
   #scaleTo: #grid: #'//' #asIntegerPoint #directionToLineFrom:to: ...)
```

- Returns all the messages the class understand

A little query

- Let us query the system... and only filter the unary messages :)

```
Point selectors select: #isUnary
> #(#x #theta #asIntegerPoint #r #negated #normalized #sign #degrees #isIntegerPoint #guarded
  #fourNeighbors #eightNeighbors #min #max #ceiling #normal #asPoint #y #abs #isPoint #angle
  #transposed #reciprocal #asFloatPoint #asNonFractionalPoint #rounded #leftRotated #floor #truncated
  #hash #deepCopy #fourDirections #rightRotated #isSelfEvaluating #asMargin #isZero)
```

- Easy :)

Binary Messages

```
anObject aBinarySelector anArgument
```

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

Binary Message Examples

```
1 + 2
> 3
```

```
2 > 3
> false
```

```
10@200
> 10@200
```

```
'Black chocolate' , ' is good'
> 'Black chocolate is good'
```

Keyword Messages

```
anObject keyword1: argument1 keyword2: argument2
```

equivalent to

```
receiver.keyword1keyword2(argument1, argument2)
```

Keyword Messages for Javaists

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

Keyword Messages for Javaists

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

Keyword Messages for Javaists

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

Keyword Messages for Javaists

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

Keyword Messages for Javaists

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

Keyword Messages for Javaists

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

postman send: mail to: recipient
```

- The message is named `send:to:`
- It is sent to `postman`
- With two arguments: `mail` and `recipient`

Message setX:

```
10@20 setX: 2
> 2@20
```

- We change the x value of the receiver (a point)

Message at:put:

```
#('Calvin' 'hates' 'Suzie') at: 2 put: 'loves'
> #('Calvin' 'loves' 'Suzie')
```

- #(...) creates an array
- `at:put:` changes the value of the array element.
- arrays start at 1 in Pharo (i.e., first element is at index 1)

Message between:and:

```
12 between: 10 and: 20
> true
```

- Here the message `between:and:` is sent to an integer
- Takes two arguments `10` and `20`

Quizz

- `1 log`
- `Browser open`
- `2 raisedTo: 5`
- `'hello', 'world'`
- `10@20`
- `point1 x`
- `point1 distanceFrom: point2`

Composition: from left to right!

- Remember: (Msg) $>$ Unary $>$ Binary $>$ Keywords
- What happens when we have two messages of the same kind?
- From left to right

```
1000 factorial class name
> LargePositiveInteger
```

is equivalent to

```
(((1000 factorial) class) name)
```

- Ease the composition of messages.

Back to Message Precedence

Remember that we have only messages

- (Msg) $>$ Unary $>$ Binary $>$ Keywords
- From left to right

Precedence Example

```
2 + 3 squared
> 2 + 9
> 11
```

- unary (squared) first then binary (`+`)

Precedence Example

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

- binary (`+`) first then keyword-based (`raisedTo:`)

Precedence Example

```
Color gray – Color white = Color black
> aGray – aWhite = aBlack
> aBlack = aBlack
> true
```

- Unary then binary from left to right

Precedence Example

```
1 class maxVal + 1
> 1073741824
```

- unary, unary and binary

```
1 class
> SmallInteger

1 class maxVal
> 1073741823

1 class maxVal + 1
> 1073741824

(1 class maxVal + 1) class
> LargePositiveInteger
```

Getting the Pharo Logo

```
(ZnEasy getPng: 'http://pharo.org/web/files/pharo.png')
   asMorph openInWindow.
```

- unary (`asZnUrl`), keyword (`getPng:`)
- then unary, unary

Parentheses take precedence!

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

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

The price for simplicity

- Only messages: `+`
  - is a message, no precedence
  - can be redefined in domain classes
- Simple
- One limit: no mathematical precedence

No mathematical precedence

```
3 + 2 * 10
> 5 * 10
> 50
```

- should be rewritten using parentheses

```
3 + (2 * 10)
> 3 + 20
> 23
```

```
1/3 + 2/3
> 7/3 /3
> 7/9
```

- should be rewritten using parentheses

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

Quiz

Describe the order in which the messages are executed

- `(10@20 corner: 100@200) topCorner`
- `10@20 distanceFrom: 200@200`
- `2 + 3 raisedTo: 3 + 2`

Message Sequence

```
message1.
message2.
message3
```

- . is a separator, not a terminator,
- No need to put one at the end.

```
| macNode pcNode node1 printerNode |
macNode := Workstation withName: #mac.
Transcript cr.
Transcript show: 1 printString.
Transcript cr.
Transcript show: 2 printString
```

Multiple messages to an object

- To send multiple messages to the same object

```
Transcript show: 1 printString.
Transcript cr
```

is equivalent to:

```
Transcript
   show: 1 printString ;
   cr
```

Imagine we want to add 2 to a new set.

```
Set new add: 2
> 2
```

- The message `add:` returns its argument and not the receiver.
- We have to use a temporary variable.

```
| s |
s := Set new
```

A puzzle

```
Set new add: 2
> 2
```

- The message `add:` returns its argument and not the receiver.
- We have to use a temporary variable.

```
| s |
s := Set new.
s add: 1; add: 2.
s
```

- What could be another solution?

One Hint

- What if we get a message that returns the receiver?

yourself

```
Object>>yourself
    ^ self
```

Solution

```
| s |
s := Set new.
s add: 1; add: 2.
s
```

is equivalent to

```
Set new add: 1; add: 2; yourself
> aSet
```

Summary

- Three kinds of messages: unary, binary and keywords
- Arguments are placed inside message structure:
  - 2 between: 0 and: 5 (the message is `between:and:`)
- `()` takes precedence over messages
- . is a separator
- ; is useful to avoid to repeat receiver