

CS 535 Object-Oriented Programming & Design  
Fall Semester, 2013  
Doc 3 Smalltalk Syntax  
Sep 3 2013

Copyright ©, All rights reserved. 2013 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/openpub/>) license defines the copyright on this document.

## References

Object-Oriented Design with Smalltalk — a Pure Object Language and its Environment, Ducasse, University of Bern, Lecture notes 2000/2001, [http://www.iam.unibe.ch/~ducasse/WebPages/Smalltalk/ST00\\_01.pdf](http://www.iam.unibe.ch/~ducasse/WebPages/Smalltalk/ST00_01.pdf)

Smalltalk Best Practice Patterns, Kent Beck, Prentice Hall, 1997

# Basic Smalltalk Syntax

The Xerox team spent 10 years developing Smalltalk

They thought carefully about the syntax of the language

Smalltalk syntax is

- Different from other languages

- Simple and compact

- Designed for readability

# The Rules

Everything in Smalltalk is an object

All actions are done by sending a message to an object

Every object is an instance of a class

All classes have a parent class

Object is the root class

# Sample Program

"A Sample comment"

| a b |

a := 'this is a string'.

":= is assignment"

a := 'this is " a string that contains  
a single quote and a newline'.

a := 'concat' , 'inate'.

a := 5.

a := 1 + "comments ignored" 1.

b := 2 raisedTo: 5.

^a + b

"^ means return"

# Multiple Assignments

Assignment statements return values!

| a b |

a := b := 3 + 4.

a and b now contain 7

# Statement Separator

```
| cat dog |  
cat := 5.  
dog := cat + 2
```

A period is used as a statement separator

A period is optional after the last statement

# Identifiers

An identifier (any name) in Smalltalk is of the form:

letter (letter | digit )\*

| cat dog |

cat := 5.

dog := cat + 2.



# Literals

String	'aString'
Character	\$a \$A
Symbol	#cat
Integer	12
Floating Point	12.4 0.123 1.567e5 1.543q-8
Array	#{1 2 3 'cat'}
Boolean	true false
Undefined	nil



# Messages

No operators in grammar

Operators are methods in classes

+ is a method in the Integer class

In  $3 + 4$ , + is a message sent to the integer 3

# Three type of Messages

## Binary

1 + 2

12 / 6

## Unary

12.3 printString

'123' asNumber

## Keyword

'Hi mom' copyFrom: 1 to: 3

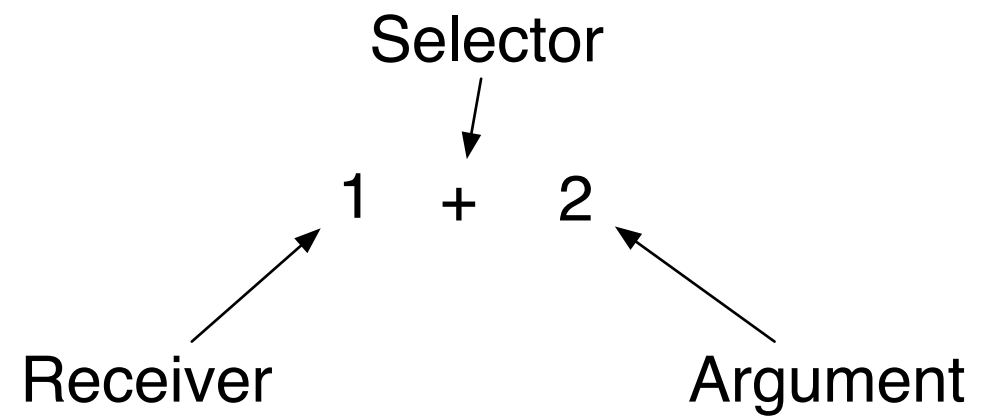
# Message Structure

All messages contain:

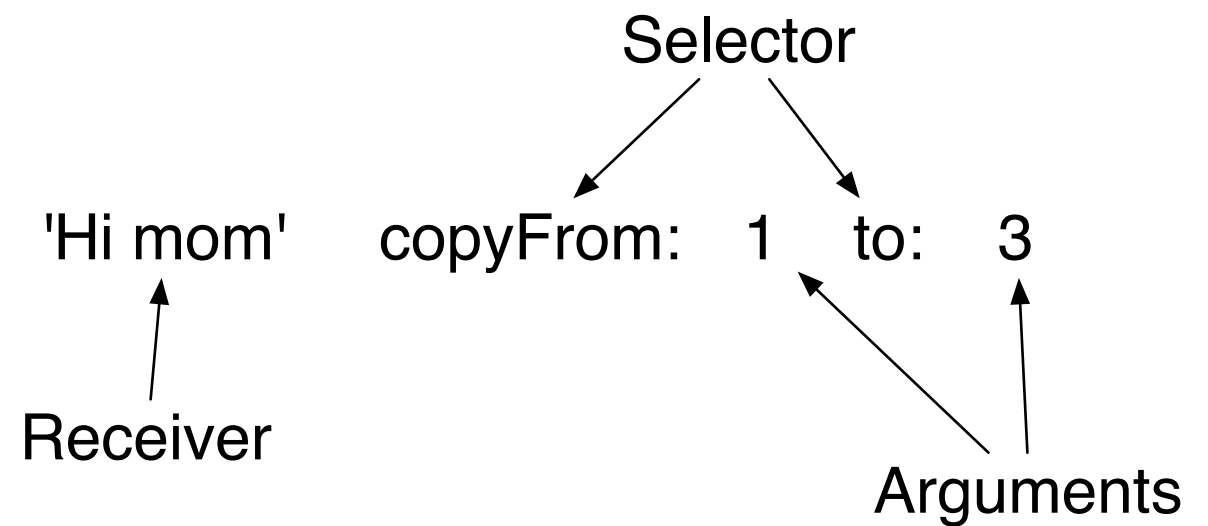
Receiver

Selector

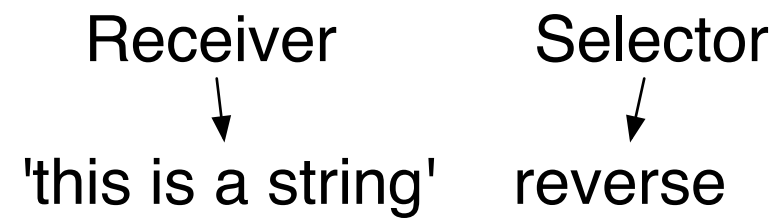
Zero or more arguments



Messages always return a value



# Unary Messages



Compared to

`"this is a string"->reverse(); //C++`

`"this is a string".reverse(); //Java`

`'this is a string' revserse "Smalltalk"`

# Combining Unary Messages

Unary messages are executed from left to right

100 factorial printString size

is done as:

((100 factorial) printString) size

# How about this?

100 factorial size

This will not work

100 factorial returns an integer

Integers do not implement a size method

Use the Smalltalk browser to see the methods in a class



# Binary Messages

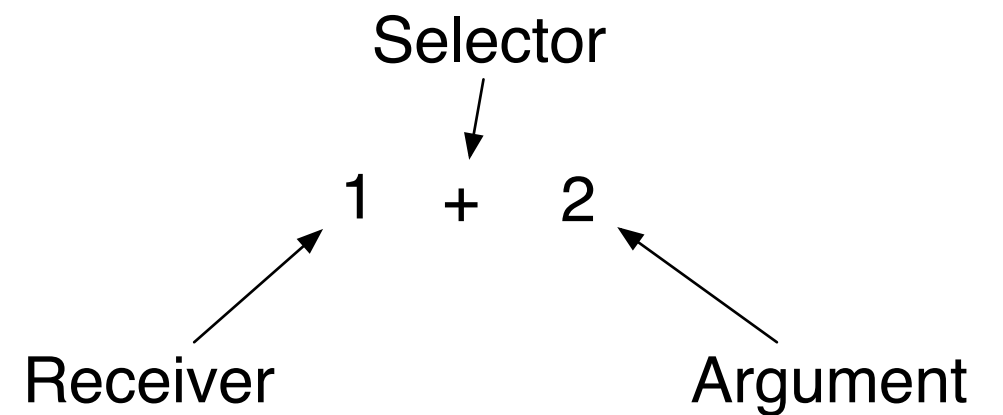
Binary selectors are

Arithmetic, comparison and logical operations

One or two characters taken from:

+ - / \ \* ~ < > = @ % | & ! ? ,

Second character is never a minus sign



# Combining Binary Messages

Binary messages are executed from left to right

$$1 + 2 * 3 * 4 + 5 * 6$$

is executed as

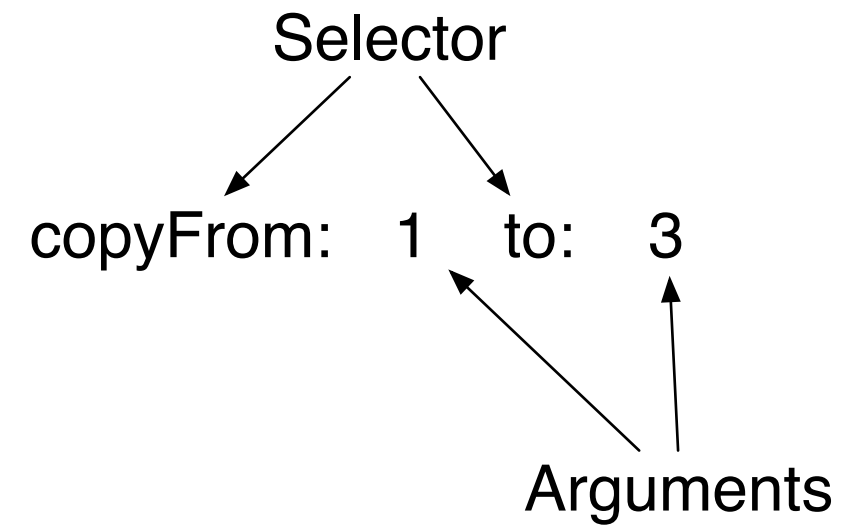
$$((((1 + 2) * 3) * 4) + 5) * 6$$

# Keyword Messages

21 min: 5.

'this is a string'  
findString: 'string'  
startingAt: 4  
ignoreCase: true  
useWildcards: false

'Hi mom'  
↑  
Receiver



Compared to

'this is a string'.find('string', 4, true, false);

'this is a string'->find('string', 4, 1, 0);

# min: implementation

min: aMagnitude

"Answer the receiver or the argument, whichever has the lesser magnitude."

self < aMagnitude

ifTrue: [^self]

ifFalse: [^aMagnitude]

# findString:startingAt:ignoreCase:useWildcards:

findString: substring startingAt: start ignoreCase: ignoreCase useWildcards: useWildcards

"Find the first occurrence of substring within the receiver.

Answer the first character to last character index range

of that occurrence. If no such match is found, answer (0 to: 0).

Begin the search at start. If ignoreCase is true, disregard

case differences. If useWildcards is true, treat # and \* specially."

^useWildcards

ifTrue: [self rangeOfPattern: substring startingAt: start ignoreCase: ignoreCase]

ifFalse:

    [] result |

    result := ignoreCase

        ifTrue: [self findSameAs: substring startingAt: start wildcard: nil]

        ifFalse: [self findString: substring startingAt: start wildcard: nil].

    result = 0

        ifTrue: [0 to: 0]

        ifFalse: [result to: result+substring size-1]].

# Keyword Messages verses Positional Argument Lists

## Keyword

Communicates role of each argument

## Positional Argument List

More common so more familiar

Easy for compiler to parse

Easier for programmer to mix up parameters

# Where do Keyword Messages End?

Compiler combines all keywords in a statement into one message

```
'this is a string'  
  copyFrom: 1  
  to: 12 min: 7
```

```
'this is a string'  
  copyFrom: 1  
  to: (12 min: 7)
```

```
copyFrom:to:min:
```

Does not exist, so results in an error

# Formatting Keyword Messages

```
'this is a string'  
  findString: 'string'  
  startingAt: 4  
  ignoreCase: true  
  useWildcards: false
```

or

```
'this is a string' findString: 'string' startingAt: 4 ignoreCase: true useWildcards: false
```

## Beck's Rule

When a keyword message has two or more keywords  
Place each keyword with its argument on its own line  
Indent the keyword one tab from the receiver



# The Tab Verses Spaces Debate

To indent a line of code do you use:

Tab

- Easier to type

- Sometimes tabs are different on screen and on hard copy

- Some companies ban tabs

Spaces

- Smalltalk handles tabs uniformly

- Use tabs to indent in Smalltalk

- Do not use spaces to indent in Smalltalk

# Precedence

First unary messages are parsed left to right

Binary messages are parsed left to right after unary messages

Keyword messages are parsed after binary messages

Parenthesis change the order of evaluation

Expression	Result
$3 + 4 * 2$	14
$3 + (4 * 2)$	11
$5 + 3 \text{ factorial}$	11
$(5 + 3) \text{ facorial}$	40320
<code>'12' asNumber + 2</code>	14

# Quiz

Parse this statement

cat cat cat: cat + cat cat: cat / cat

# Transcript

Special output window

Similar to Java's System.out and C++'s out

Transcript clear.

Transcript show: 'This is a test'.

Transcript cr.

Transcript show: 'Another line'.

Transcript tab.

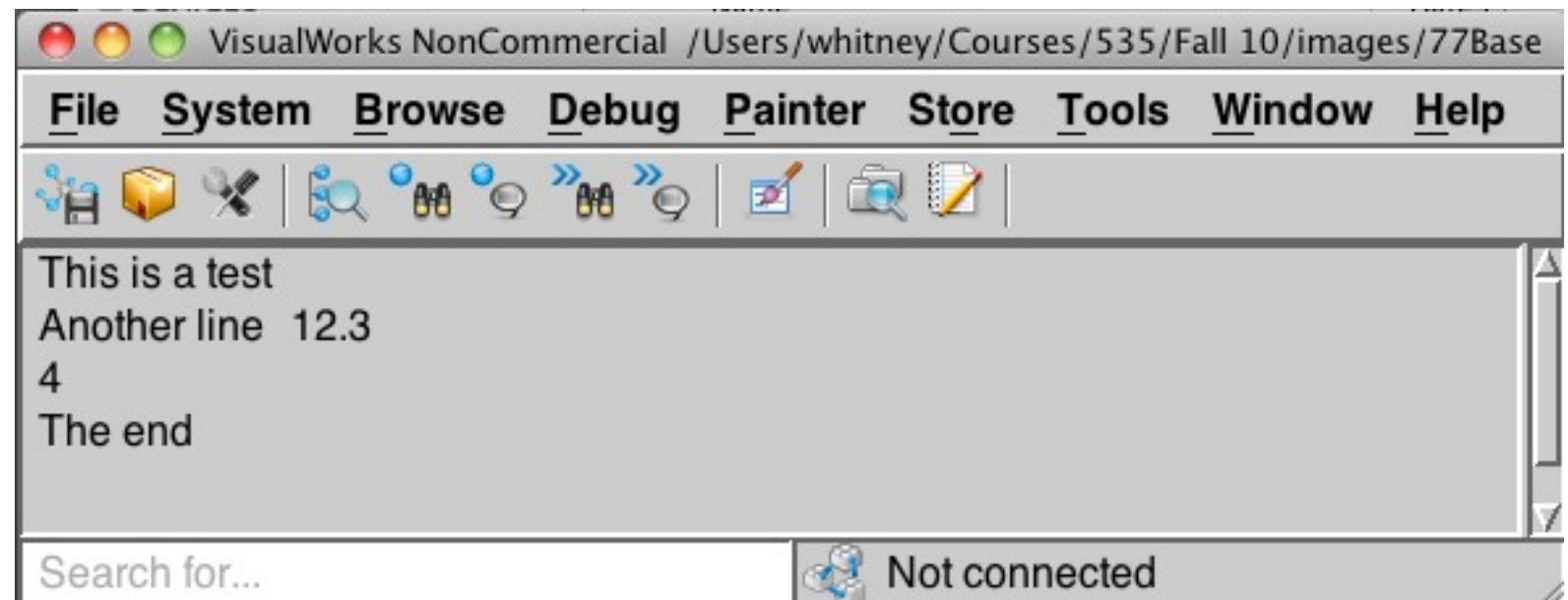
Transcript print: 12.3.

Transcript cr.

Transcript show: 4 printString.

Transcript cr.

Transcript show: 'The end'.



# Useful Transcript Methods

clear

clear the Transcript

show: aString

display aString in the Transcript

print: anObject

display a string representation of anObject in the Transcript

nextPutAll: aString

add aString to the display buffer

flush

put contents of display buffer in Transcript

empty the buffer

tab cr space crtab crtab: anInteger

put given character in the display buffer

# Cascading Messages

A cascade sends multiple messages to the same receiver

Messages are sent from left to right to the same receiver

Transcript

```
clear;  
show: 'This is a test';  
cr;  
show: 'Another line';  
tab;  
print: 12.3;  
cr;  
show: 4 printString;  
cr;  
show: 'The end'.
```

# Cascade Versus Compound Messages

'hi mom' reverse asUppercase

'MOM IH'

'hi mom' reverse; asUppercase

'HI MOM'