CS 535 Object-Oriented Programming & Design
Fall Semester, 2011
Doc 3 Smalltalk Syntax
Sep 1 2011

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

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

Reading

Smalltalk by Example, Alex Sharp, Chapter 2 Messages
PDF available from http://stephane.ducasse.free.fr/FreeBooks.html

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

There are a few exceptions to these rules. See if you can find them as we go through Smalltalk.

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.
```


Wednesday, August 31, 2011

Vertical bars at the top of a program declare variables. Variables must be declared. All variables are references to objects. Variables are initialized to nil.
As we will see numbers in Smalltalk are objects. Internally references to objects require pointers. Always using a pointer to refer to a number would slow arithemitic operations. Most Smalltalk virtual machines will store numbers directly in a variable. At the programming level one does not see any difference in how numbers and other objects are handled.

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

Integer Size

1000 factorial

4023872600770937735437024339230039857193748642107146325437999104299385123986290205920442084869694048004799886101971960586316668729948085589013
2382966994459099742450408707375991882362772718873251977950595099527612087497546249704360141827809464649629105639388743788648733711918104582578
36478499770124766328898359557354325131853239584630755574091142624174743493475534286465766116677973966688202912073791438537195882498081268678383
7455973174613608537953452422158659320192809087829730843139284440328123155861103697680135730421616874760967587134831202547858932076716913244842
6236131412508780208000261683151027341827977704784635868170164365024153691398281264810213092761244896359928705114964975419909342221566832572080
82133318611681155361583654698404670897560290095053761647584772842188967964624494516076535340819890138544248798495995331910172335555660213945039
97362807501378376153071277619268490343526252000158885351473316117021039681759215109077880193931781141945452572238655414610628921879602238389714
76088506276862967146674697562911234082439208160153780889893964518263243671616762179168909779911903754031274622289988005195444414282012187361745
9926429565817466283029555702990243241531816172104658320367869061172601587835207515162842255402651704833042261439742869330616908979684825901254
5832716822645806652676995865268227280707578139185817888965220816434834482599326604336766017699961283186078838615027946595513115655203609398818
0612138558600301435694527224206344631797460594682573103790084024432438465657245014402821885252470935190620929023136493273497565513958720559654
2287497740114133469627154228458623773875382304838656889764619273838149001407673104466402598994902222217659043399018860185665264850617997023561
93897017860040811889729918311021171229845901641921068884387121855646124960798722908519296819372388642614839657382291123125024186649353143970137
42853192664987533721894069428143411852015801412334482801505139969429015348307764456909907315243327828826986460278986432113908350621709500259738
9863554277196742822248757586765752344220207573630569498825087968928162753848863396909959826280956121450994871701244516461260379029309120889086
9420285106401821543994571568059418727489980942547421735824010636774045957417851608292301353580818400969963725242305608559037006242712434169090
0415369010593398383577793941097002775347200000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000

Wednesday, August 31, 2011

Integers are as large as you have memory to hold them

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

Using messages rather than hard coded grammar makes parsing code simpler and language extensible

Three type of Messages

Binary

```
1 + 2
12 / 6
```

Unary

```
12.3 printString
'123' asNumber
```

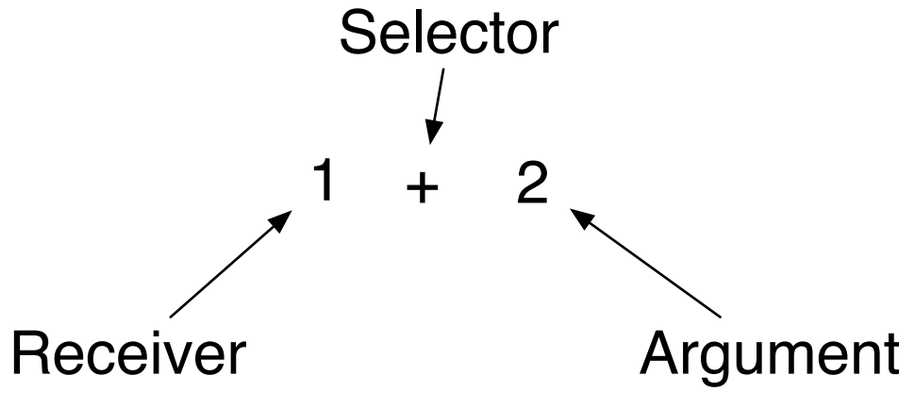
Keyword

```
'Hi mom' copyFrom: 1 to: 3
```

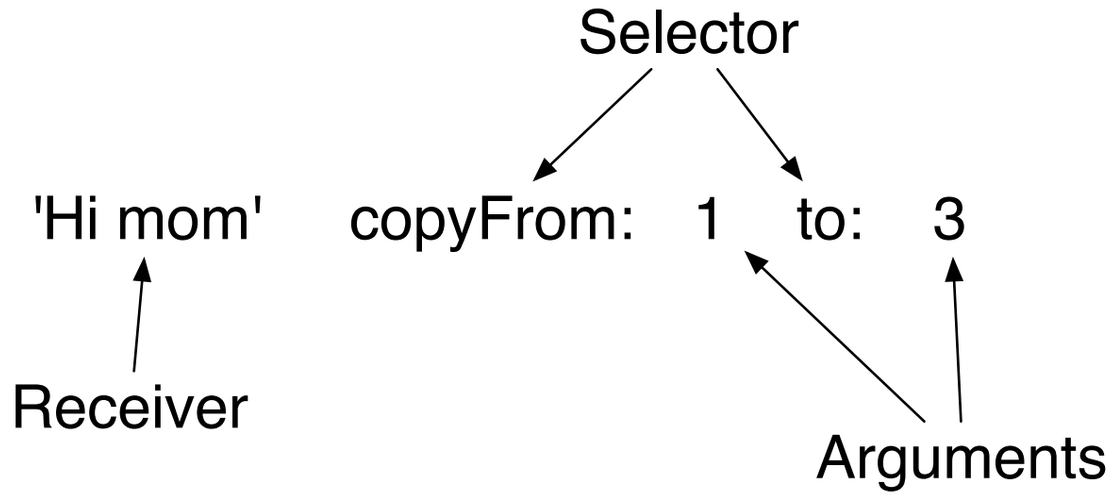
Message Structure

All messages contain:

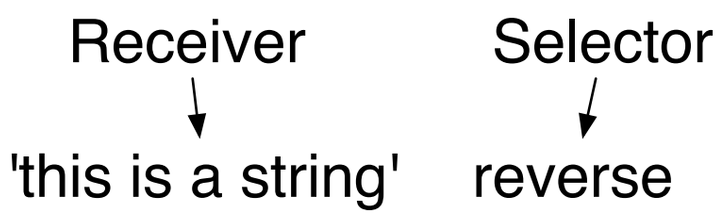
- Receiver
- Selector
- Zero or more arguments



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

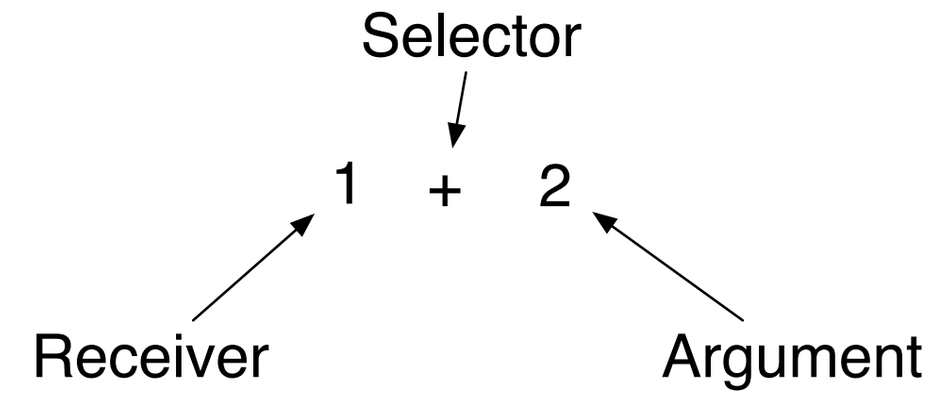
This will not work

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

Using the above rules you can create your own binary messages in Smalltalk. You can make @? a binary method in a class.

Combining Binary Messages

Binary messages are executed from left to right

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

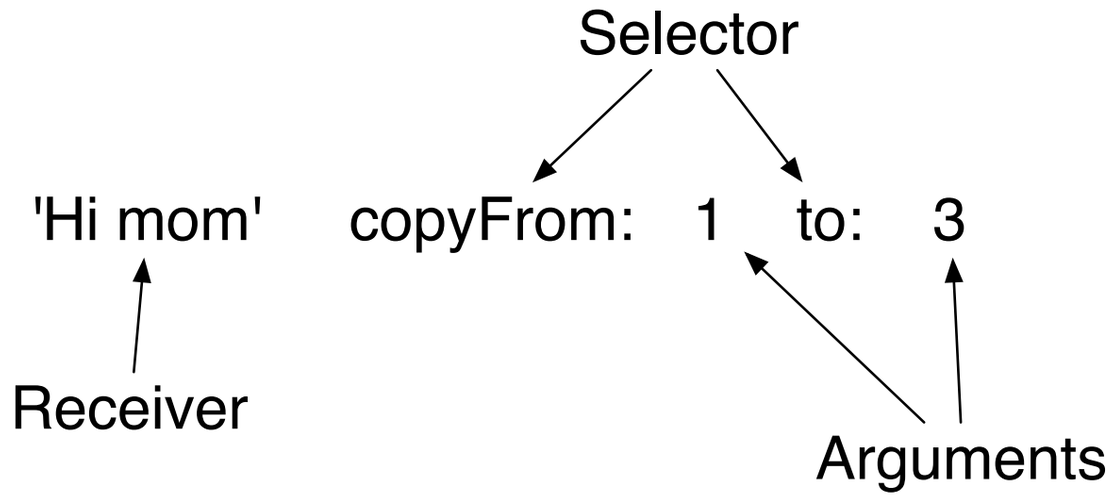
is executed as

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

Keyword Messages

21 min: 5.

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



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

copyFrom:to:min:
```

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

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

Or you can just have the browser format your code for you.

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

Or you can have the browser format your code.

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 factorial	11
(5 + 3) facorial	40320
'12' asNumber + 2	14

Quiz

Parse this statement

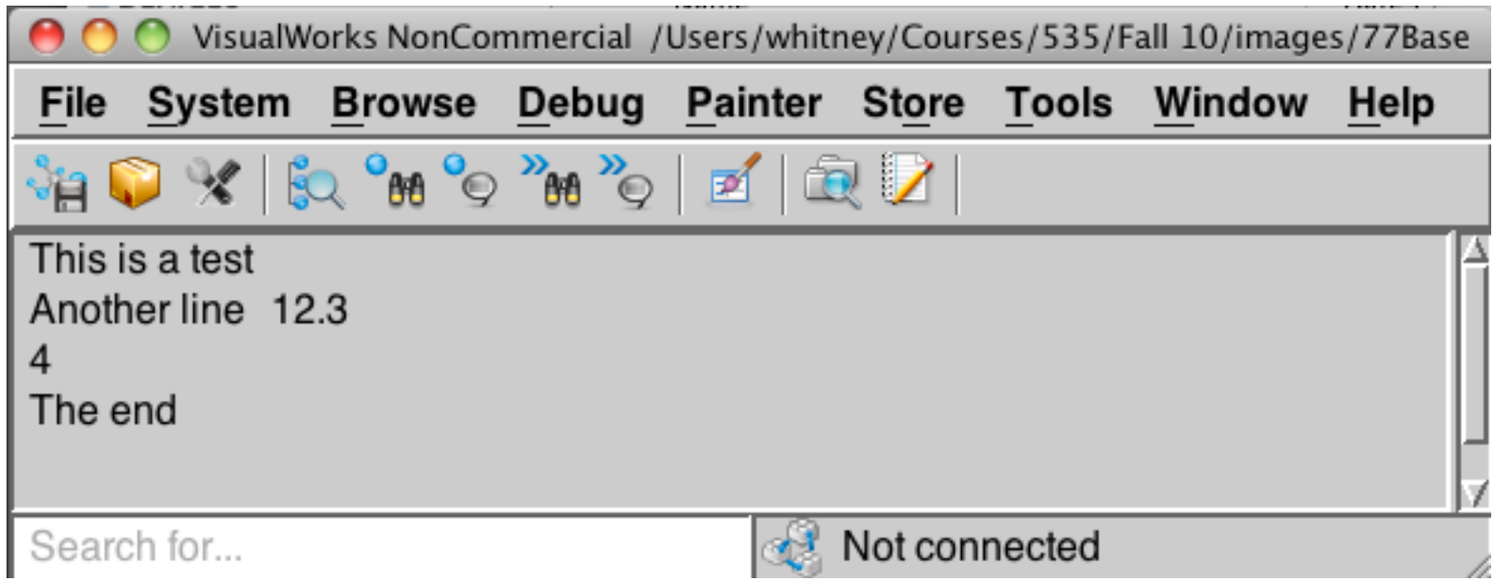
cat cat cat: cat + cat cat: cat / cat

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'