Smalltalk in a Nutshell

- OO Model in a Nutshell
- Syntax in a Nutshell

Smalltalk OO Model

- ***Everything*** is an object
  - Only message passing
  - Only late binding
- Instance variables are private to the object
- Methods are public
- Everything is a pointer
- Garbage collector
- Single inheritance between classes

Power & Simplicity: The Syntax on a PostCard

```
exampleWithNumber: x
"A method that illustrates every part of Smalltalk method syntax except
  primitives. It has unary, binary, and key word messages, declares arguments
  and temporaries (but not block temporaries),  accesses a global variable (but
  not and instance variable), uses literals (array, character, symbol, string,
  integer, float), uses the pseudo variable true false, nil, self, and super, and has
  sequence, assignment, return and cascade. It has both zero argument and one
  argument blocks. It doesn't do anything useful, though"
  |y|
  true & false not & (nil isNil) ifFalse: [self halt].
  y := self size + super size.
  #($a #a 'a' 1 1.0)
          do: [:each | Transcript
                          show: (each class name);
                          show: (each printString);
                          show: ' '].
  ^ x < y
```

Language Constructs

^	return
"	comments
#	symbol or array
'	string
[ ]	block or byte array
.	separator and not terminator (or namespace access in VW7)
;	cascade (sending several messages to the same instance)
|	local or block variable
:=	assignment
$	character
:	end of selector name
e, r	number exponent or radix
!	file element separator
<primitive: ...>	for VM primitive calls

Syntax

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

float: 1/33

boolean: true, false

point: 10@120

Note that @ is not an element of the syntax, but just a message sent to a number. This is the same for /, bitShift, ifTrue:, do: ...

Syntax in a Nutshell (II)

assigment: var := aValue
block: [:var ||tmp| expr...]

temporary variable:	|tmp|
block variable:	:var
unary message:	receiver selector
binary message:	receiver selector argument
keyword based:	receiver keyword1: arg1 keyword2: arg2...
cascade:	message ; selector ...
separator:	message . message
result:	^
parenthesis:	(...)

Messages instead of a predefined Syntax

- In Java, C, C++, Ada constructs like >>, if, for, etc. are hardcoded into the grammar
- In Smalltalk there are just messages defined on objects
- (>>) bitShift: is just a message sent to numbers
  - 10 bitShift: 2
- (if) ifTrue: is just messages sent to a boolean
  - (1> x) ifTrue:
- (for) do:, to:do: are just messages to collections or numbers
  - #(a b c d) do: [:each | Transcript show: each ; cr]
  - 1 to: 10 do: [:i | Transcript show: each printString; cr]
- -> Minimal parsing
- -> Language is extensible

Class and Method Definition Revisited

- Class Definition: A message sent to another class

```
Object subclass: #Tomagoshi
        instanceVariableNames: 'tummy hunger
dayCount'
        classVariableNames: ''
        poolDictionaries: ''
        category:  'Monster Inc'
```

- Instance variables are instance-based protected (not visible by clients)

Method Revisited

- Normally defined in a browser or (by directly invoking the compiler)
- Methods are public

```
Tomagoshi>>digest
"Digest slowly: every two cycles, remove one from the
  tummy"

  (dayCount isDivisibleBy: 2)
        ifTrue: [ tummy := tummy -1]
```

Instance Creation

- 1, 'abc'

- Basic class creation messages are

  new, new:,

  basicNew, basicNew:

  Monster new

- Class specific message creation (messages sent to classes)

  Tomagoshi withHunger: 10