GNU Smalltalk (1)

gst > 'Hello World' printNL

(a) stdout << 'Hello World' << Character nl.

script #! ~~~~ /gst

3+4 | 3 is the receiver object
+4 is the message
+ is the selector

class vs instance

1 class — unary selector/message
1 + 2 — binary selector (+) message (+2)
1 from: 3 to: 8 — keyword selector from:to:
argument 2 — arguments 3, 8

'Hello World' size
3 bitAt: 3 put: 1 ⟹ 7
selector bitAt:put:

unary - alphanumeric
binary - non-alp but max 2 chars
Keyword - variadic, each selector end in :

precedence: 1st unary
2nd binary
3rd keyword.

3+4*5
(3+4)*5

( ) override
same prec. strict L → R

Message chaining

foo bar baz qux

send bar to foo; send baz to result; then qux that 2nd result

expr: 1 + 2 * 3 + 4 * 5

value: 1 → 3 → 9 → 13 → 65

Message cascading

foo bar; baz; QUX

- sends messages in sequence all to foo

Classes

ex: String
SmallInteger
Number

Object

'foo' class
36 class

3 -8 16.4 1e2

8r312 16rABC

(6/3) (2/4) — fractions

-j8 -j1 — imaginary

Reserved words
nil
true
false
self
super

operators + - * /

// int div
\\ int modulus

quo:
rem:

i between: m and: n ⟹ m ≤ i ≤ n

n abs
18 cos (radians)
n negated
n raisedTo: a

n squared
n even
n odd

Smalltalk ③

Numbers ...

```
n  sign
n  integerPart
n  truncated
n  rounded
(3/4)  denominator
(3/4)  numerator
```

Character

```
$a  $% $$
c asLowercase
c asUppercase
c isAlphaNumeric
c isDigit
```

$10' → $

String

```
'abcd'
'foo' includes: $o
'Canol' indexOf: $n
        ↪ 3
'ab','cd' ⇒ concat
'abcd' size
```

Variables

```
|x| — declares x
x := 6 — assignment
```

Arrays

```
a := Array new: 10
        — get 10 nil values —
a at: i — subscript
a at: i put: n
a reverse
a includes: n ⟶ true or false
```

Set

```
s := Set new
s add: 'foo'
s remove: 'foo'
```

Dictionary

```
a := Dictionary new
a at: 'foo' put: 'bar'
a keys ⟹ returns a Set
a removeKey: k
```

Control flow

Block is [stmt. stmt. stmt ....]

a block is an object

```
b := [3+7]
b value ⟶ 10
b2 := [:a :b | a + b]
b2 value:3 value:4 ⟹ 7
```

selection

```
b ifTrue: [...]
b ifFalse: [...]
b ifTrue: [...] ifFalse: [...]
```

repetition

```
b whileTrue: [...]
[expr] whileTrue: [...]
to:do:      1 to: 5 do: [:x ...]
to:by:do:   1 to: 100 by: 3 [...]
```

Smalltalk (5)

Inheritance (single)
- due to duck typing doesn't need Java interface
- objs & classes respond to messages

Polymorphism: (universal) overriding

```
Supclass subclass: Subclsname [
    | instancevar iv ivv |
    classv := nn .
    classvar := nn .
    Subclsname class >> clsmethod : param [
        {
            ^retobj
        }
    ]
    instmethod [
        {
            ^retobj
        }
    ]
]
```

```
foo := subclsname new.
```

← inherited from Object

can send inst methods to objs
class classes
not vice versa

```
Object subclass: Animal [
    |name|
    new: n [name:=n]
    name [^name]
]
Animal subclass: Cat [
    speak [^'meow']
]
Animal subclass:Lion [
    speak [^'roar']
]
```

```
b := Cat new: 'Bastet'.
s := Lion new: 'sekhmet'.
stdout << b name << ': ' << b speak << nl.
stdout << s name << ': ' << s speak << nl.
```

Self and super

self ≡ this in Java

super ≡ super in Java

Reading

"Compute Programming using GNU Smalltalk"
by Carol Gökel
(114 pages)