The Minimal Squeak Introduction

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.                                          " := is assignment"
a := 1 + "comments ignored" 1.

b := 2 raisedTo: 5.
↑ a + b                                          " ↑ means return"
                                                 " ^ means return"
```

← is the '_' character (same as :=)
↑ is the '^' character (same as ^)

Characters

```
| aChar |
aChar := $a.
aChar := $5.
aChar := Character tab.
aChar := Character value: 65.
aChar := 65 asCharacter.
```

Numbers

Integer arbitrary precision
Float Range ±10^307
Fraction Operations remain as a fraction
(1/3)*(1/2) ==> (1/6)
((1/3)*(1/2)) asFloat ==> 0.16666666666666666

Literal forms

1234
12345678901234567890123456789012345678901234567890123456
12.34
123e2
3.14e-10

<base>r<number>
16rFF
2r1010e2

Symbols

Special type of strings

All symbols comprising the same characters have the same instance

#'A string preceded with a hash sign is a Symbol'
#orAnyIdentifierPrefixedWithAHashSymbol

Messages

All messages:

- Are sent to an object
- Return a value

Binary

2 + 4

2 is the receiver

+ is the binary messsage

4 is the argument

returns 6

Unary

12 factorial

12 is the receiver

factorial is the message

returns 479001600

'this is a string' reversed

returns 'gnirts a si siht'

'this is a string' asUppercase reversed

returns 'GNIRTS A SI SIHT'

'12345' isAllDigits

returns true

Keywords

12 min: 6

12 is the receiver

min: is the message

6 is the argument

returns 6

'this is a string' copyFrom: 1 to: 7

'this is a string' is the receiver

copyFrom:to: is the message

1 and 7 are the arguments

returns 'this is'

'this is a string'
findString: 'string'
startingAt: 4
caseSensitive: true

Precedence

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

```
3 + 4 * 2           =   14
3 + (4 * 2)         =   11
5 + 3 factorial     =   11
(5 + 3) factorial   =   40320
```

```
'this is a string' reversed
    findString: 'string'
    startingAt: 1 + 2
    caseSensitive: 2 + 2 = 4
```

Parenthesis must be used to separate multiple keyword messages in one statement

```
'this is a string' reversed
    findString: ('the cat is white' copyFrom: 9 to: 10)
    startingAt: 1 + 2
    caseSensitive: 2 + 2 = 4
```

Multiple Assignment statements

Assignment statements return values!

```
| a b |
a := b := 3 + 4.
```

a now contains 7

Transcript

Special output window

Useful messages:

open
    open the window

clear
    clear the window

show: aString
    display aString in the window

nextPutAll: aString
    add aString to the display buffer

endEntry
    put contents of display buffer in window
    empty the buffer

tab cr space crtab crtab: anInteger
    put given character in the display buffer

Sample Program

```
Transcript open.
Transcript show: 'This is a test'.
Transcript cr.
Transcript show: 'Another line'.
Transcript cr.
Transcript show: 'The end'.
```

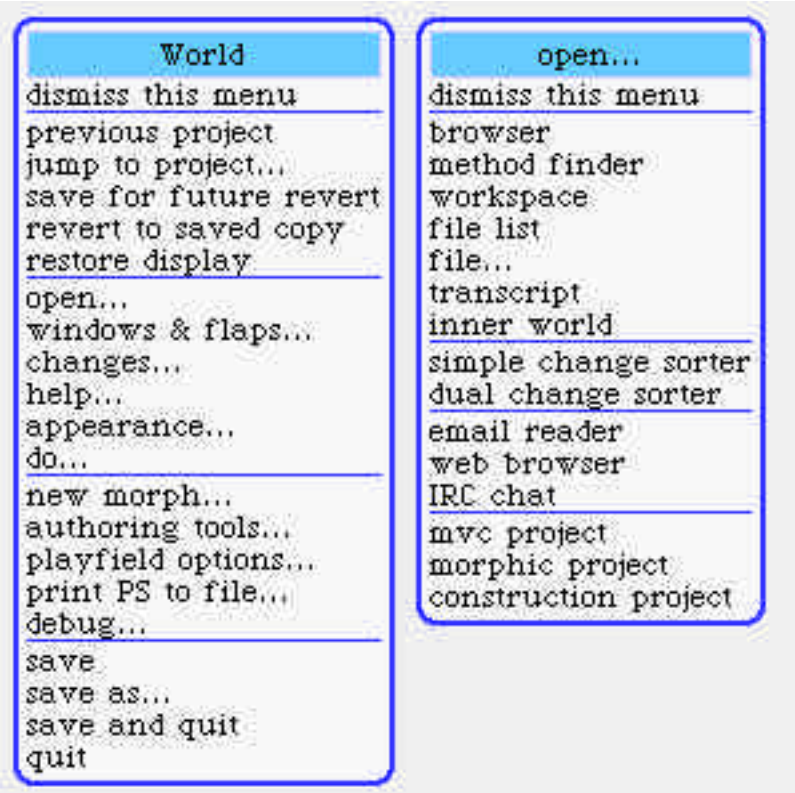
Cascading Messages

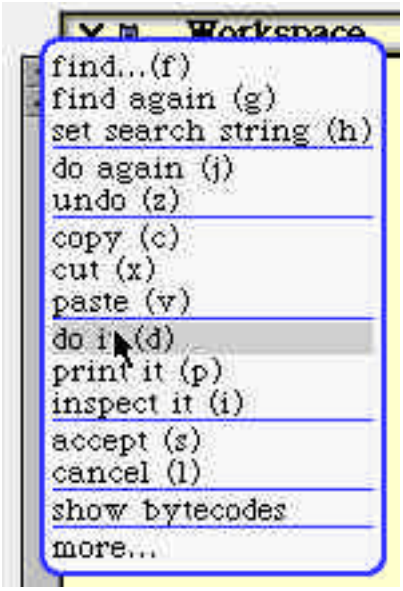
```
Transcript
    open;
    show: 'This is a test';
    cr;
    show: 'Another line';
    cr;
    show: 'The end'.
```

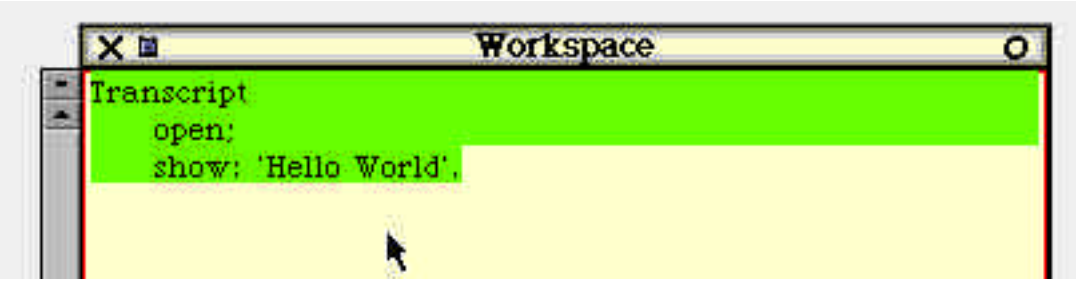
List messages to send to the same receiver

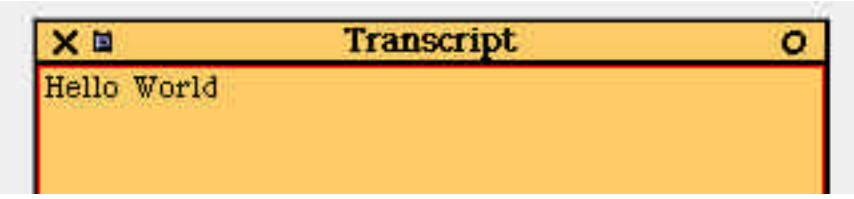
Separate the messages with ';'

Some Squeak Environment

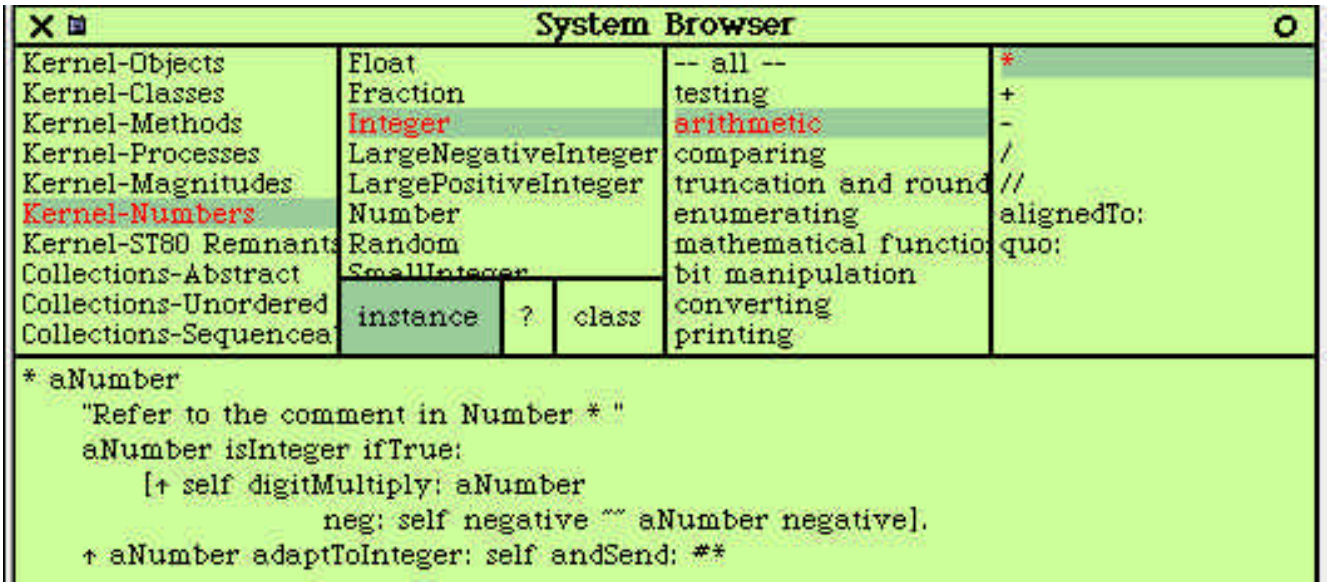








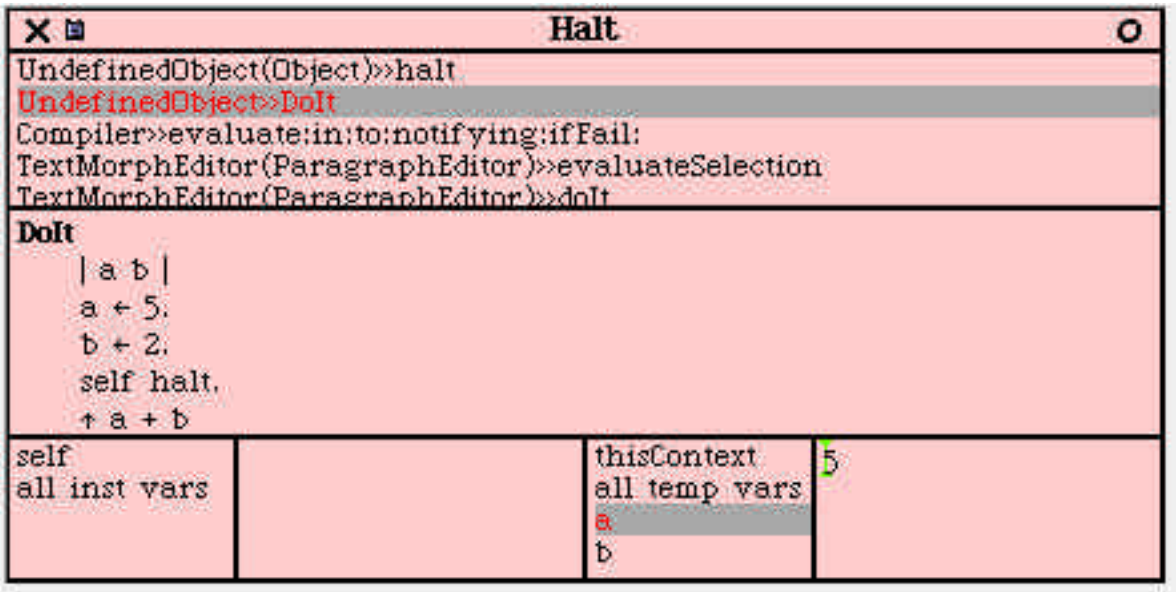
System Browser



halt

opens the debugger on code

```
| a b |
a := 5.
b := 2.
self halt.
^a + b
```



Smalltalk Term	C++ Equivalent
self	this
super	super

Blocks

A block:

- is a function object
- can have 0 or more arguments
- is executed when sent the message 'value'

Zero Argument Block

```
| block x |
x := 5.
block := [Transcript show: x printString].
x := 10.
block value
```

Prints 10 in the Transcript window

```
| block x |
x := 5.
block := [:argument | Transcript show: (x + argument) printString].
x := 10.
block value: 4
```

Blocks and Return Values

Blocks return the value of the last executed statement in the block

```
| block x |
block := [:a :b |
                    | c |
                    c := a + b.
                    c + 5].

x := block value: 1 value: 2.
```

x has the value 8

Control Structures

IF

```
| diff |
diff := (x > y)
            ifTrue: [ x - y]
            ifFalse: [ y - x]
```

Variations

```
ifTrue: [code]

ifFalse: [code]

ifFalse: [code1] ifTrue: [code2]
```

while

```
| x y difference |
x := 8.
y := 6.
difference := 0.
[x > y] whileTrue:
      [difference := difference + 1.
      y := y + 1].
^difference

| count |
count := 0.
[count := count + 1.
 count < 100] whileTrue.
Transcript clear; show: count printString
```

Variations

```
whileFalse: [code]
whileFalse
```

Boolean values

true
Unique instance of class True

false
Unique instance of the class False

ifTrue:ifFalse, etc are messages sent to true, false

nil

Value of an uninitialized variable

Object

All 'things' in Smalltalk are objects

Objects are created from classes

The class Object is the parent class of all classes

Contains common methods for all objects

printString	isNil	==
class	notNil	=
halt	~~	~=

Equality

All objects are allocated on the heap

Variables are references (like a pointer) to objects

A == B
Returns true if the two variables point to the same location

A = B
Returns true if the two variables point to objects that logically represent the same object.

In Smalltalk you want to use '=' nearly all the time

A ~= B
(A = B) not

A ~~ B
(A == B) not

```
| a b c |
a := #(1 2 3 ).
b := a copy.
c := #(1 2 3).
a = b                   "true"
a == b                  "false"
a == c                  "true       smart compiler"
```

Enumeration Control Structures

```
Transcript
	open;
	clear.
3 timesRepeat:
	[Transcript
		cr;
		show: 'Testing!'].

1 to: 3 do:
	[ :n |
	Transcript
		cr;
		show: n printString;
		tab;
		show: n squared printString].

9 to: 1 by: -2 do:
	[ :n |
	Transcript
		cr;
		show: n printString].
```

Collections

Array
OrderedCollection growable array
String
Intervals
Set order not important
Bag set with item frequency

Arrays
Literal

Literal arrays

- are created compile time
- intially contains only other literals
- indexing starts at 1

```
| array |
array := #( 1 2 'cat' ).          "literal array"
array
      at: 1
      put: 'dog'.
^array
```

returns ('dog' 2 'cat')

Some Array methods

at:	size
at:put:	copyFrom: index1 to: index2
sort	includes: anObject
reversed	isEmpty

Dynamic Array Creation

```
|  array x |
x := 'cat'.
array := { 1. 'mouse'.  x }.
array
      at: 1
      put: 'dog'.
^array

returns ('dog' 'mouse' 'cat' )

|  array x |
x := 'cat'.
array := Array
                  with: 'dog'
                  with: 'mouse'
                  with: x.

^array

|  array  |
array := Array new: 10.
1 to: 10 do:
      [ :n |
      array
            at: n
            put: n].
array := array shuffled.        "Creates new array in random order"
^array
```

OrderedCollection

Growable arrays
Used far more often than arrays

```
| list |
list := OrderedCollection new: 3.
1 to: 10 do: [ :n | list add: n].
list := list shuffled.
^list
```

returned OrderedCollection (6 4 5 3 2 9 10 7 1 8)

Enumerating

do: aBlock	Evaluate aBlock with each of the receiver's elements as the argument.
select: aBlock	Evaluate aBlock with each of the receiver's elements as the argument. Collect into a new collection like the receiver, only those elements for which aBlock evaluates to true. Answer the new collection.
reject: aBlock	Evaluate aBlock with each of the receiver's elements as the argument. Collect into a new collection like the receiver only those elements for which aBlock evaluates to false. Answer the new collection.
collect: aBlock	Evaluate aBlock with each of the receiver's elements as the argument. Collect the resulting values into a collection like the receiver. Answer the new collection.
detect: aBlock	Evaluate aBlock with each of the receiver's elements as the argument. Answer the first element for which aBlock evaluates to true. Signal an Error if none are found.
inject: initialValue into: binaryBlock	Accumulate a running value associated with evaluating the argument, binaryBlock, with the current value of the argument, thisValue, and the receiver as block arguments.

Examples

Code	Return value
#(1 2 3 4 5 6) collect: [:each | each squared]	(1 4 9 16 25 36)
'this is a string' select: [:each | each isVowel]	'iiai'
#(1 7 2 3 9 3 50) detect: [:each | each > 8]	9
#(1 7 2 3 9 3 50) reject: [:each | each even]	(1 7 3 9 3)

C++ like Code

```
| data sum |
data := #( 1 7 2 3 9 3 50).
sum := 0.
1 to: data size do: [:each | sum := sum + (data at: each) squared].
sum
```

Redone with do:

```
| sum |
sum := 0.
#( 1 7 2 3 9 3 50) do: [:each | sum := sum + each squared].
sum
```

Redone the Smalltalk way

```
#( 1 7 2 3 9 3 50) inject: 0 into: [:sum :each | sum + each squared]
```

Streams

```
| a |
a := WriteStream on: (String new: 10).
a
	nextPutAll: 'cat';
	cr;
	nextPutAll: 'dog'.
a contents


| aFile |
aFile := FileStream fileNamed: 'test'.
aFile
	nextPutAll: 'cat';
	cr;
	nextPutAll: 'dog';
	close.

aFile contentsOfEntireFile
```

Classes

Types of Variables

Instance Variable
- Like protected C++ data member
- Accessible only by instances (objects) of the class

Class Variable
- Like protected static C++ data member
- Accessible the class and instances (objects) of the class

Class Instance Variable
- No C++ equivalent
- Accessible only by the class

Pool Variable
- No C++ equivalent
- Accessible by class and instances
- Can be shared by other class
- A bit complex

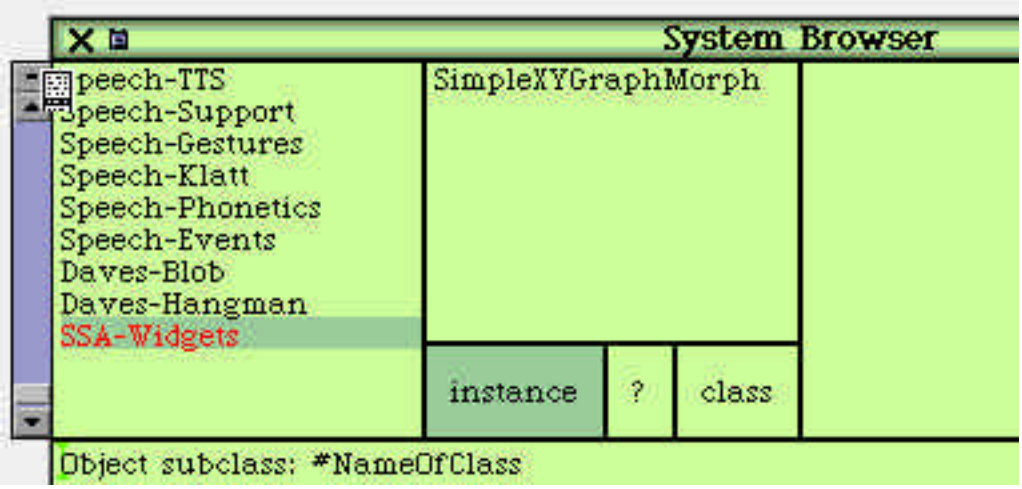
Types of Methods

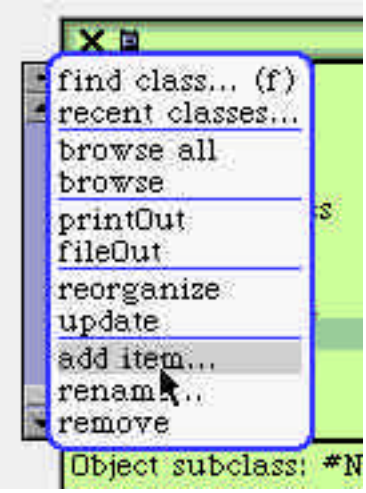
Instance methods
- Sent to instances of Classes

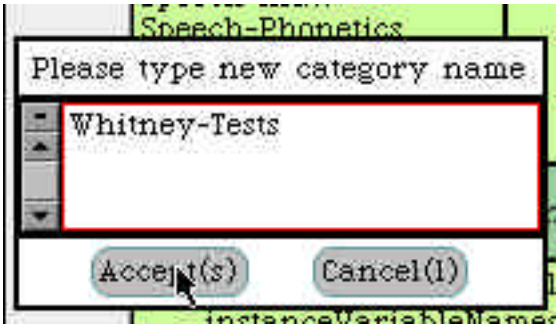
Class methods
- Sent to Classes

Creating Classes

Step one: Create a category for the class in the System Browser(if it needs a new category



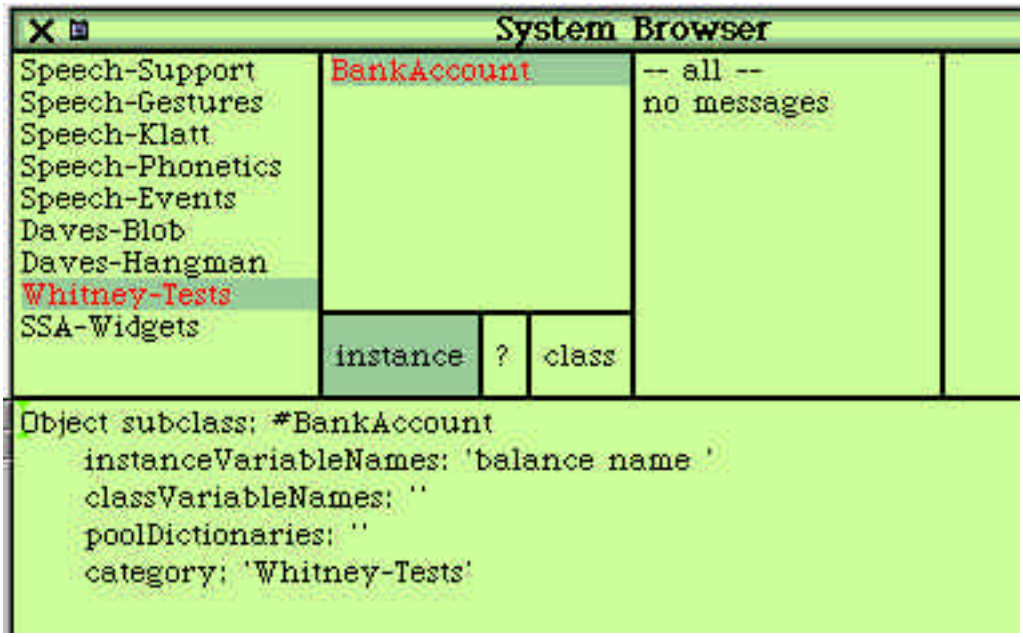




Step 2. Create the class
Edit the class template in the lower pane of the browser.
Then accept (or save) the changes via the pane menu or command-s

The template is:
Object subclass: #NameOfClass
instanceVariableNames: 'instVarName1 instVarName2'
classVariableNames: 'ClassVarName1 ClassVarName2'
poolDictionaries: ''
category: 'Whitney-Tests'

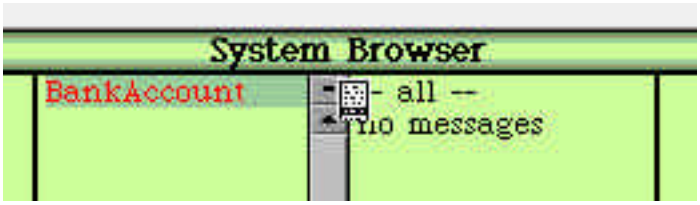
After the save we have:

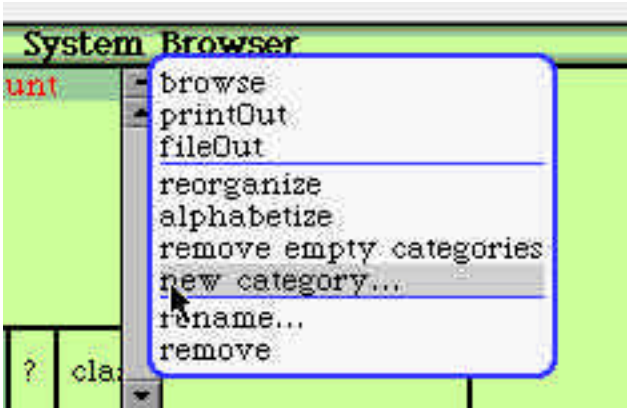


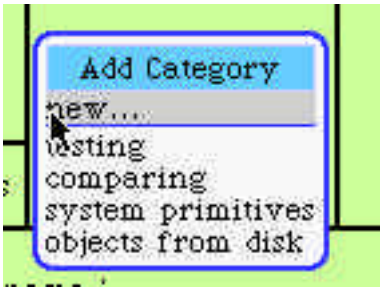
Adding a method

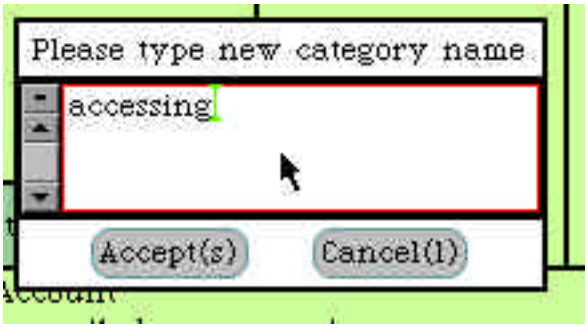
Select a method category (third upper pane) or create one if it does not exist

Creating a method category



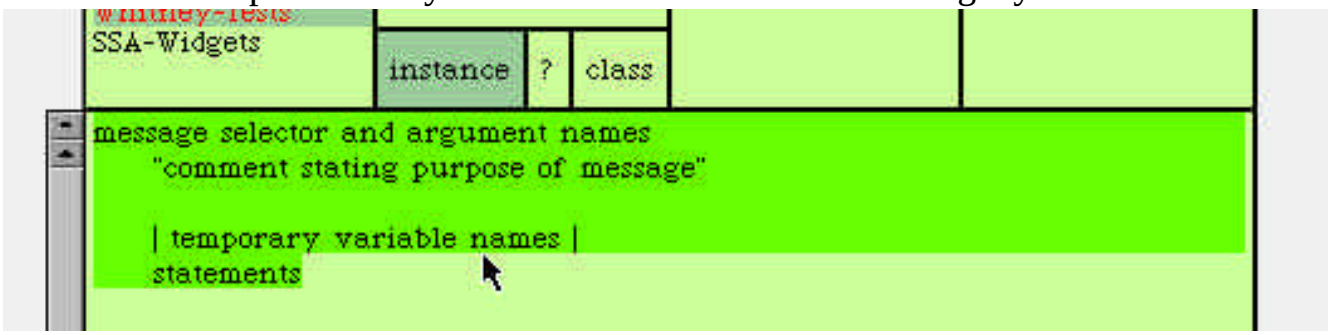


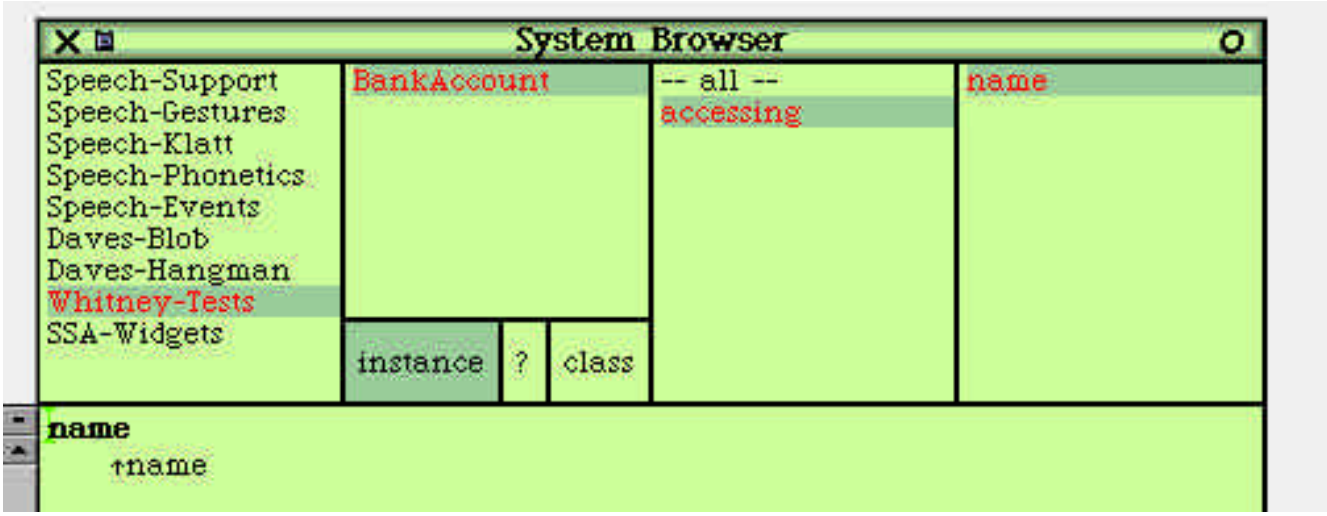




Adding the method

Edit the lower pane once you have selected a method category






```
Object subclass: #BankAccount
	instanceVariableNames: 'name balance '
	classVariableNames: ''
	poolDictionaries: ''
	category: 'Whitney-Examples'

initializeName: aString
	name  aString

printOn: aStream
	aStream
		nextPutAll: 'Account(';
		nextPutAll: name;
		nextPutAll: '-';
		nextPutAll: balance;
		nextPut: $)

deposit: aNumber
	aNumber < 0
		ifTrue:[self error: 'Attempt to deposit negative amount'].
	balance  balance + aNumber

name
	^name

withdrawl: aNumber
	aNumber < 0
		ifTrue:[self error: 'Attempt to withdrawl negative amount'].
```

```
        balance - aNumber > 0
            ifTrue:[self error: 'Attempt to withdrawl more than current balance'].
        balance  balance - aNumber

Class method
name: aString
        ^super new initializeName: aString
```