Cincom Smalltalk™ – The Language on Two Pages

Language and Environment

- Smalltalk is a language and an environment to use the language. This sheet focuses on the language element.
- Everything is an object. Every object is an instance of a class which defines the behavior of the object.
- Classes inherit from class **Object**, using single inheritance.
- One does things by sending a message to an object. If the message is understood by the object, then it has a matching method which it executes.
- Objects have instance variables that can only be accessed by the methods of the object. All methods are public to all objects.
- Methods can have temporary variables that exist only for the execution of the method. For example the variable newSelf is declared and assigned as follows:

```
changeCapacityTo: newCapacity
    | newSelf |
    newSelf := self copyEmpty: newCapacity
```

- **nil** is the unique instance of the class **UndefinedObject** and is the default value of a variable which has had no explicit value assigned.
- **super** is used to invoke the superclass' implementation of a method.
- The boolean values true and false are single instances of the classes **True** and **False**.
- Some objects are literal types: **Integer** (123), **Float** (123.4), **Character** ($a), **String** ('abc'), **Symbol** (#abc) and **Array** (#(123 123.4 $a 'abc' #abc)) when all its elements are literals.

Method Basics

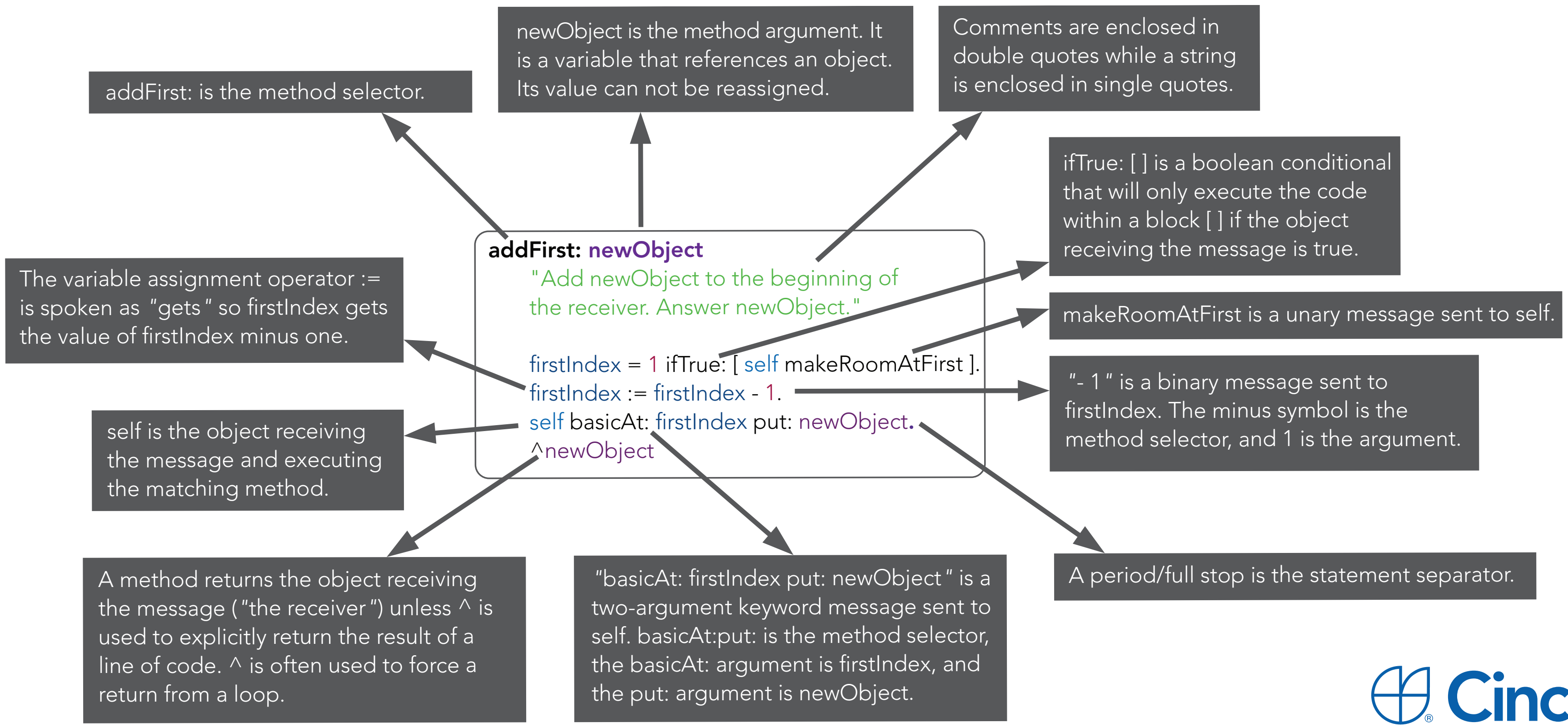
Execution order is evaluated left to right until the statement separator (a period/full stop: .) is reached. Everything within parentheses ( ) is evaluated first, with the contents of the inner-most parentheses evaluated first. Messages are evaluated as follows:

All **unary messages**, those with no arguments, are evaluated first.

Then all **binary messages**, those with one argument whose method selector does not end in a colon and is one or more non-alphanumeric symbols.

Then **keyword messages**, which take one or more arguments and use a word with a colon before each argument.

Anatomy of a Method



```
addFirst: newObject
    "Add newObject to the beginning of
    the receiver. Answer newObject."

    firstIndex = 1 ifTrue: [ self makeRoomAtFirst ].
    firstIndex := firstIndex - 1.
    self basicAt: firstIndex put: newObject.
    ^newObject
```

addFirst: is the method selector.

newObject is the method argument. It is a variable that references an object. Its value can not be reassigned.

Comments are enclosed in double quotes while a string is enclosed in single quotes.

ifTrue: [ ] is a boolean conditional that will only execute the code within a block [ ] if the object receiving the message is true.

makeRoomAtFirst is a unary message sent to self.

"- 1" is a binary message sent to firstIndex. The minus symbol is the method selector, and 1 is the argument.

The variable assignment operator := is spoken as "gets" so firstIndex gets the value of firstIndex minus one.

self is the object receiving the message and executing the matching method.

A method returns the object receiving the message ("the receiver") unless ^ is used to explicitly return the result of a line of code. ^ is often used to force a return from a loop.

"basicAt: firstIndex put: newObject" is a two-argument keyword message sent to self. basicAt:put: is the method selector, the basicAt: argument is firstIndex, and the put: argument is newObject.

A period/full stop is the statement separator.

Cincom Smalltalk™ – The Language on Two Pages

Block Basics

These are called anonymous or lambda functions in other languages.

[1 + 2] is a Block. The simple way to get it to execute is to send it the value message.

```
[ 1 + 2 ] value.                "returns 3"
[ :x | x + 2 ] value: 1.        "returns 3"
```

A two block argument block

```
[ :x :y | x + y ] value: 1 value: 2. "returns 3"
```

Processes are a good example of block usage:

```
[ (Delay forSeconds: 5) wait.
Transcript show: 'done' ] fork.
```

Streams

WriteStream is used to write a sequence of objects to a collection.

```
writeStream := WriteStream on: Array new.
writeStream nextPut: 'Once'.
        "returns 'Once' "
writeStream nextPutAll: #( $a 42 2003 ).
        "returns #($a 42 2003)"
writeStream contents.
        "returns #('Once' $a 42 2003)"
```

ReadStream is used to read a sequence of objects from a collection.

```
readStream :=
        'Once upon a time' readStream.
readStream next.           "returns $O"
readStream upTo: $o.
        "returns 'nce up' "
readStream skip: 2.
readStream peek.           "returns $a"
readStream upToEnd.        "returns 'a time' "
readStream atEnd.          "returns true"
```

Boolean Behavior

The boolean values true and false are single instances of the classes **True** and **False**.

They are the building blocks of conditional and looping program execution. You can ask a range of questions of something and get an answer true or false such as:

```
true not.                   "returns false"
```

and you can ask several questions:

```
1 even or: [ 2 odd ].       "returns false"

23 < 25 and: [ 26 > 14 ].   "returns true"
```

You can then do something if those questions are true or false:

```
1 = 1 ifTrue: [ 'equal' ].      "returns 'equal' "
1 = 1 ifFalse: [ 'unequal' ].  "returns nil"
(10 / 2) isInteger ifTrue: [ 'integer' ]
         ifFalse: [ 'fraction' ].  "returns 'integer' "
```

Booleans can control looping:

```
i := 1.
[ i > 10 ] whileFalse: [ i := i * 2 ].
```

The first block is evaluated and if the result is false the second block is evaluated and then the loop starts again. whileTrue: also exists.

Fixed Iteration

```
10 timesRepeat: [ Transcript show: 'ping' ;
         cr ].
1 to: 10 do: [ :index |
         Transcript show: index printString; cr ].
```

You can also create an infinite loop by sending a block the message repeat. This can be escaped from by pressing **Control** + **Y**.

Collections

The Collection hierarchy provides a fundamental set of classes that group objects together. These include String, Array, OrderedCollection and Dictionary.

An **Array** is a fixed length Collection where each slot has an automatic integer based key. A **String** is an Array of Characters. An **OrderedCollection** is an expandable version of Array. A **Set** has no order and no duplicates. A **Dictionary** allows you to define unique keys and access its contents via those keys.

```
alphabet := 'abcdefghijklmnopqrstuvwxyz'.
vowels := nil.
upperVowels := nil.
firstVowel := nil.
aSentence := 'This is going to change.'.
oc := OrderedCollection new.

vowels := alphabet select: [ :letter | letter isVowel ]. "returns 'aeiou' "
upperVowels := vowels collect: [ :letter | letter asUppercase ]. "returns 'AEIOU' "
firstVowel := alphabet detect: [ :letter | letter isVowel ] ifNone: [ nil ]. "returns $a"
aSentence := aSentence , ' But not by much' .
"comma is the concatenation method. The expression returns
'This is going to change. But not by much' "
aSentence findString: 'going' startingAt: 1. "returns 9"
aSentence includes: $e. "returns true"
aSentence contains: [ :each | each isLowercase ]. "returns true"
aSentence endsWith: 'change.'. "returns false"
(aSentence allSatisfy: [ :each | each isLowercase ])
    ifFalse: [ aSentence := aSentence asLowercase ].
"returns 'this is going to change. but not by much' "
alphabet do: [ :letter | oc add: letter ].
"returns alphabet, and oc now contains each letter in a slot"
oc at: oc size. "returns $z"
oc removeLast. "returns $z, and oc has shrunk by one slot"
oc addLast: aSentence. "adds slot at end with content
'this is going to change. but not by much' "

#(5 4 2 6) inject: 0 into: [ :each :result | each + result ].
"returns 17, (5+4+2+6). The first time the block is called result gets the value 0
(it is 'injected' into the block) and then the block iterates over the Array with
result getting the value of the previous block execution each time"
```

Questions?
Check out www.cincomsmalltalk.com
or email eurosmalltalk@cincom.com.