

Class syntax

```
MyClass = SuperClass (
    | field1 field2 field3 |

    aUnaryMethod = ( ... )
    <<< argument = ( ... )
    oneArgumentMethod: argument = ( ... )
    twoArgumentMethod: argument1 otherArgument: argument2 = ( ... )

    ----

    new = ( ... )
    new: argument = ( ... )
)
```

Method body basics

```
method: argument = (
    | local1 local2 |
    ...
)
```

- Basic keywords and syntax:

self factorial: i = (... self factorial: (i-1) ...)

super testCondition = (super testCondition && ...)

assignment localOrField := 87

sequencing doThis . doThat

return identity: i = (^i)

comments "This is a comment"

literals 87, #symbol, 'This is a string'

- Useful basic messages:

object instantiation MyClass new

operators + - < = || && (and so on)

Blocks and control structures

blocks I [...body...] (lexical scoping, LIFO)

conditional (1<2) ifTrue: [...] ifFalse: [...]

looping [i>0] whileTrue: [i := i-1] (also whileFalse:)

blocks II [:parameter | ... body ...] (any number)

iteration vector do: [:element | element println]

Protocol for Object

`class` get class of object

`=` value equals

`==` primitive equals

`isNil` is this object nil? (yes, `Nil` is an object)

`asString` string conversion

, Vector construction

`value` evaluate (interesting for blocks)

`print,println` printing

`error:` error reporting

`subClassResponsibility` used for abstract classes

`doesNotUnderstand:arguments:` error handling; can do interesting things

...