JS

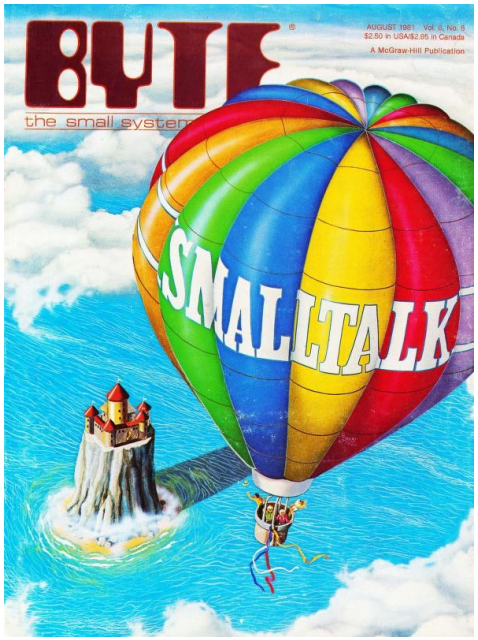
JavaScript

JavaScript (JS) is a lightweight interpreted or JIT-compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles. Read more about JavaScript.

ES

ECMAScript

The standard for JavaScript is ECMAScript. As of 2012, all modern browsers fully support ECMAScript 5.1. Older browsers support at least ECMAScript 3. On June 17, 2015, ECMA International published the sixth major version of ECMAScript, which is officially called ECMAScript 2015, and was initially referred to as ECMAScript 6 or ES6. Since then, ECMAScript standards are on yearly release cycles. This documentation refers to the latest draft version, which is currently ECMAScript 2018.



Smalltalk y Javascript

Dos lenguajes OO bien particulares

Herramientas diferentes para pensar diferente

- Presentar los aspectos más particulares de dos lenguajes que se diferencian del resto
- Smalltalk
  - OO de pies a cabeza (escrito en Smalltalk)
  - Su ambiente que invita a un enfoque exploratorio de desarrollo
  - Fuente de muchas de las ideas que hoy vemos en otros lenguajes y ambientes
- JavaScript (ECMAScript)
  - Su naturaleza basada en prototipos

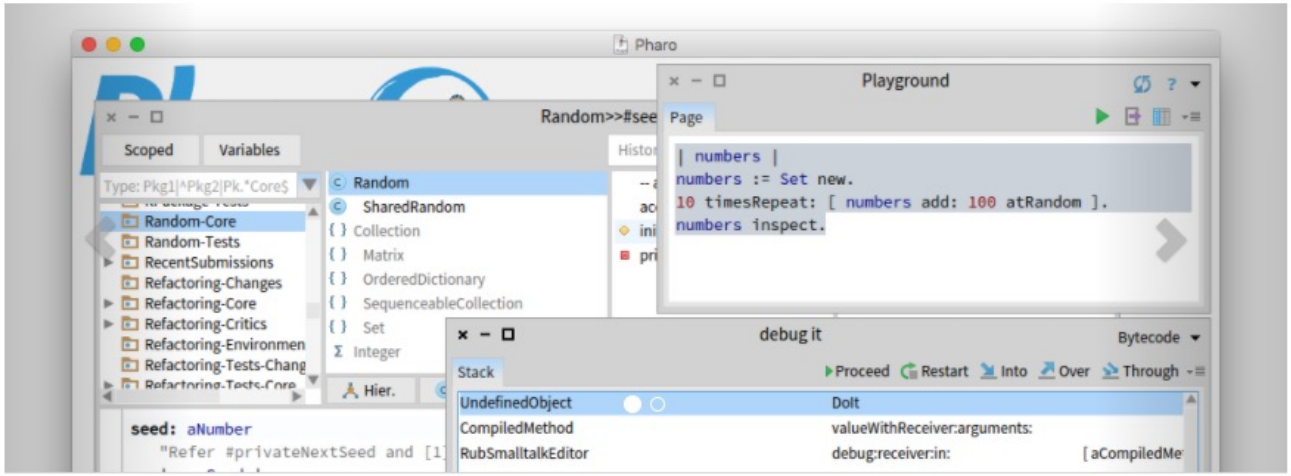
Smalltalk

- Lenguaje OO puro – todo es un objeto (¡ incluso las clases !)
- Tipado dinámicamente
- Propone una estrategia exploratoria (construccionista) al desarrollo de software
- El ambiente es tan importante como el lenguaje
  - Está implementado en Smalltalk
  - Ricas librerías de clases (fuentes de inspiración y ejemplos)
  - Todo su código fuente disponible y modificable
  - Tiene su propio compilador, debugger, editor, inspector, perfilador, etc.
  - Es extensible
- Sintaxis minimalista (con sustento en su foco educativo)
- Fuente de inspiración de casi todo lo que vino después (en OO)

Pharo

The immersive programming experience

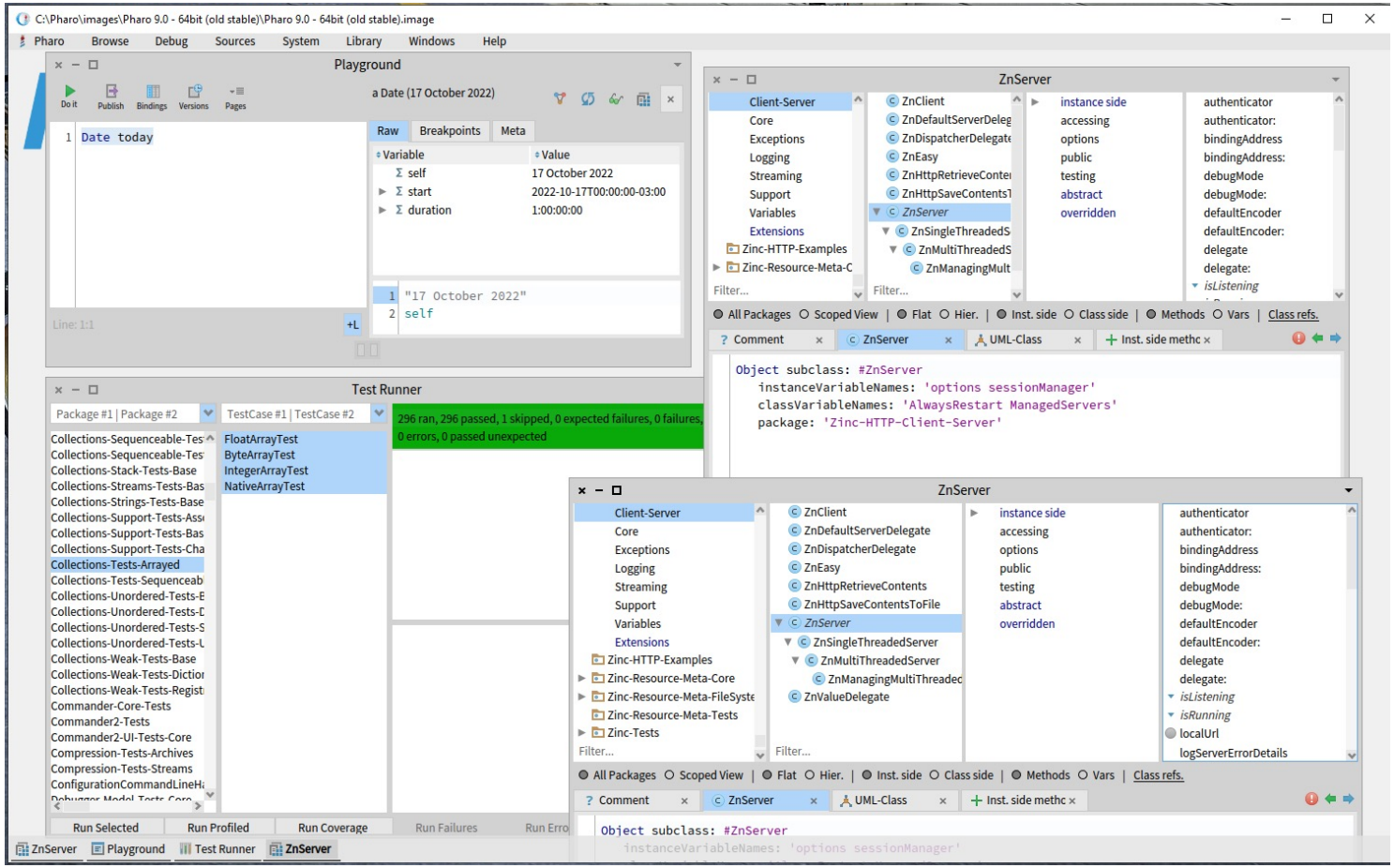
Pharo is a pure object-oriented programming language *and* a powerful environment, focused on simplicity and immediate feedback (think IDE and OS rolled into one).



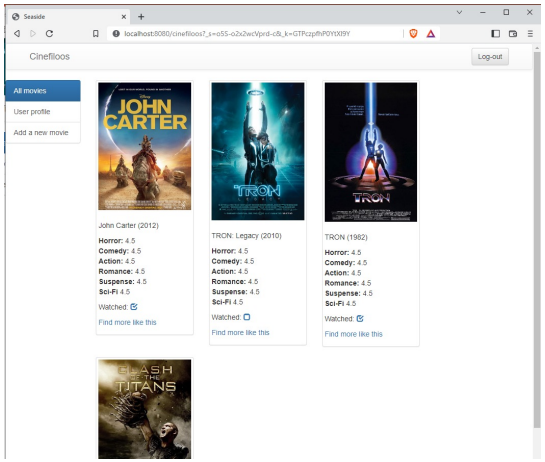
Discover | Download | Learn

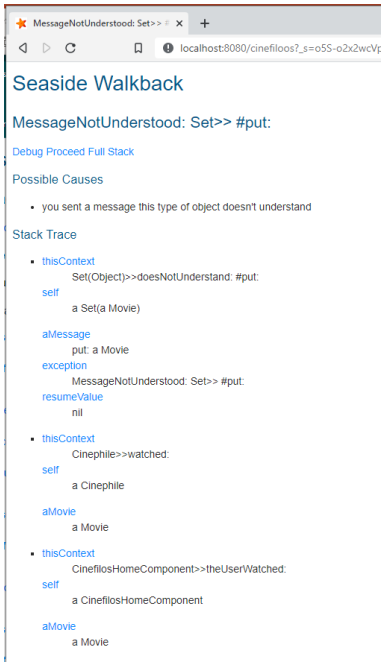
```
"Asignación y terminación"
dificil := false .

"Mensajes unarios; solo el objeto receptor, sin parámetros"
3 squared .
'Hola' reversed .

"Mensajes binarios; objeto receptor y un parámetro"

'Hola', ' Manola' .
1 @ 10 .

"Mensajes de palabra clave; objeto receptor y n parámetros"

'Manola' includesSubstring: 'ola' .
'Hola' copyWithoutAll: 'ol'.
3 between: 1 and: 2 .
```

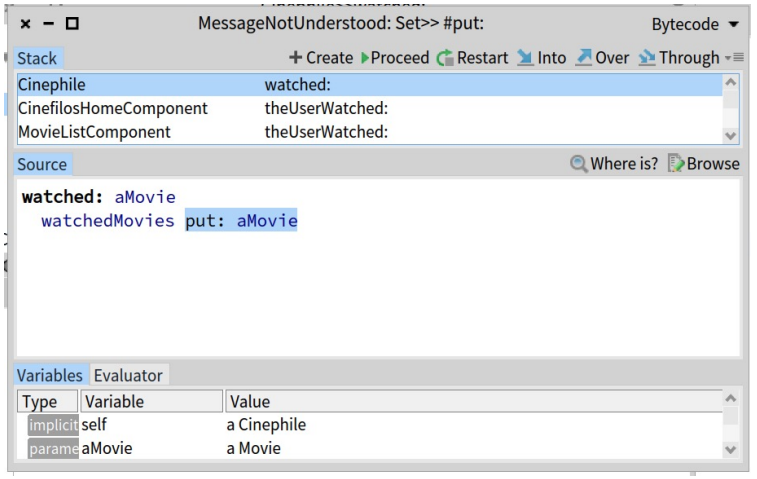
Smalltalk: el ambiente



Hot repair … (objetos vivos, siempre)







Clausuras como objetos (Closures)

```
nombre := 'Juan Gomez'.
aBlockClosure :=  [ Transcript show: 'Hola ', nombre; cr ].
aBlockClosure value.

button :=   PluggableButtonMorph  new .
button label: 'Click me'.
button position: 400@10.
button actionBlock: aBlockClosure.
button openInWorld .
```

Qué devuelven …

```
[ 1 < 10 ] value.
```

```
aBlockClosure := [ | temp |
                   temp := OrderedCollection new.
                   temp add: (Random new next).
                   temp  ].
aBlockClosure value.
aBlockClosure value.
```

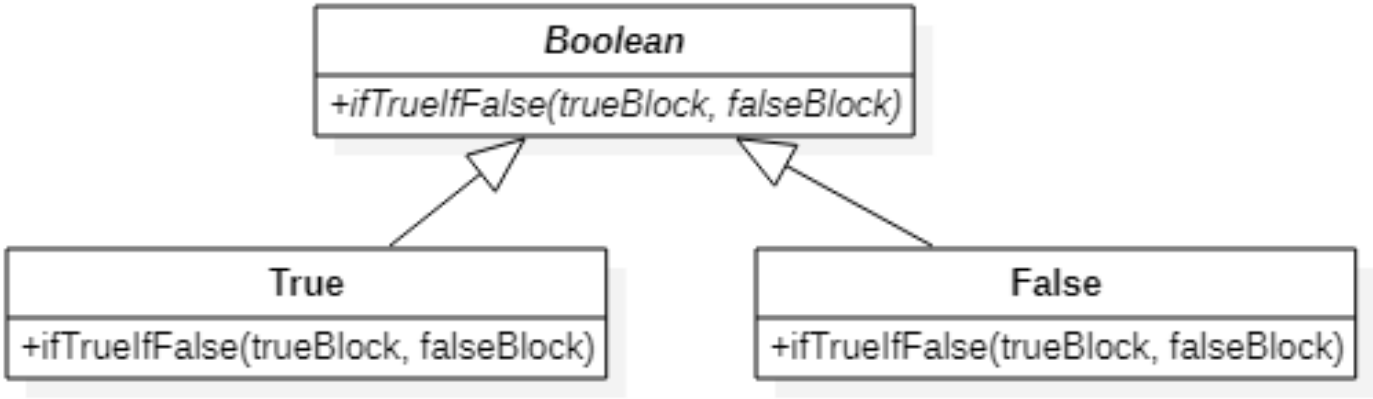
Con parámetros

```
aBlockClosure := [ :algo | algo size ].
aBlockClosure value: 'hola'.
aBlockClosure value: Set new.
```

```
aBlockClosure := [ :a :b | a < b ].
aBlockClosure value: 1 value: 2.
```

IF con objetos (puro polimorfismo)

```
(a < 100)
      ifTrue:  [Transcript shown: 'a es MENOR a 100']
      ifFalse: [Transcript shown: 'a es MAYOR a 100'  ]
```

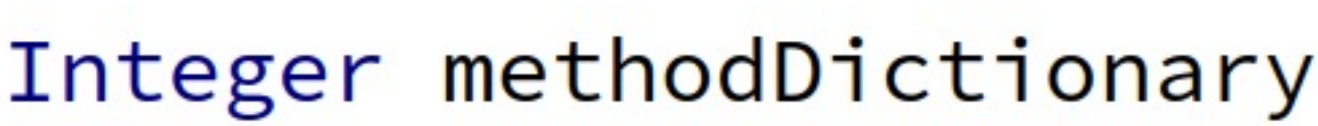


Smalltalk – las clases son objetos ...

- Smalltalk hay dos tipos de objetos: los que pueden crear instancias (de si mismos), y los que no.
  - A los primeros les llamamos clases.
- Si las clases entienden mensajes, tienen su propio conocimiento y comportamiento
  - ¿Dónde se especifica su estructura y comportamiento? ¿En otra clase?
- Esto (El metamodelo de Smalltak) en uno de sus aspectos más interesantes y desafiantes

Clases

- Hay objetos capaces de crear instancias y describir su estructura comportamiento: **las clases** (p.e., SmallInteger).
- Todo objeto es instancia de una clase (p.e., 1 de SmallInteger)
- Cuando un objeto (una instancia de una clase) recibe un mensaje, se busca el método que corresponda en **un diccionario de métodos** que mantiene la clase

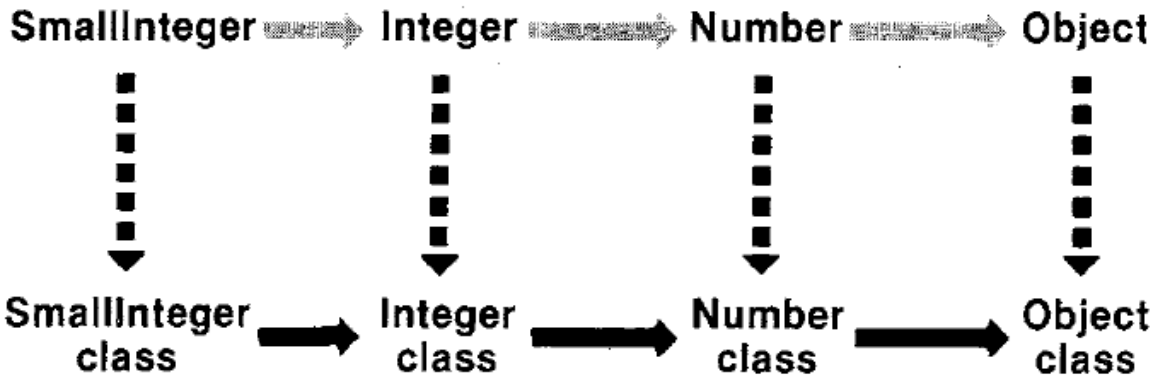



```
3 factorial .
```

Mensajes de “instancia”

Metaclases

- Las clases son instancias de una clase también (su metaclase).
- Por cada clase hay una metaclase (se crean juntas).
  - SmallInteger es instancia de “SmallInteger class”
  - Object es instancia de “Object class”
  - ...
- La metaclase de Object **(Object class)** tiene el diccionario donde se buscan los mensajes que recibe Object

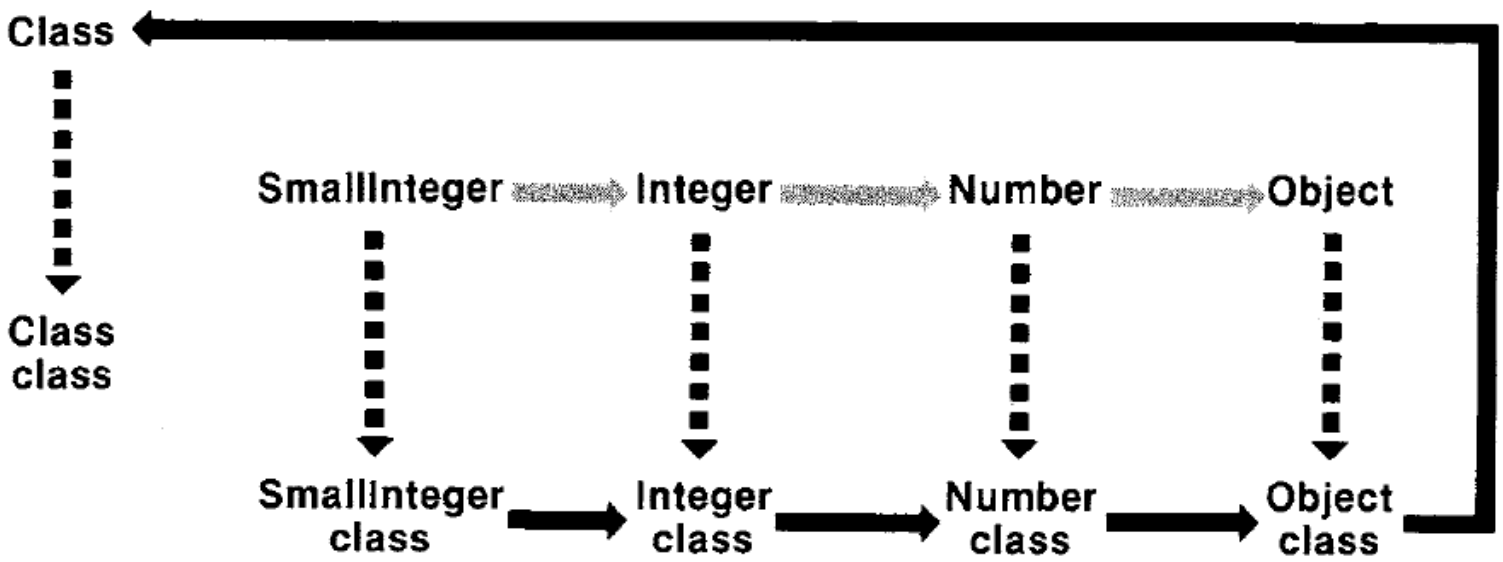



```
SmallInteger maxVal .
```

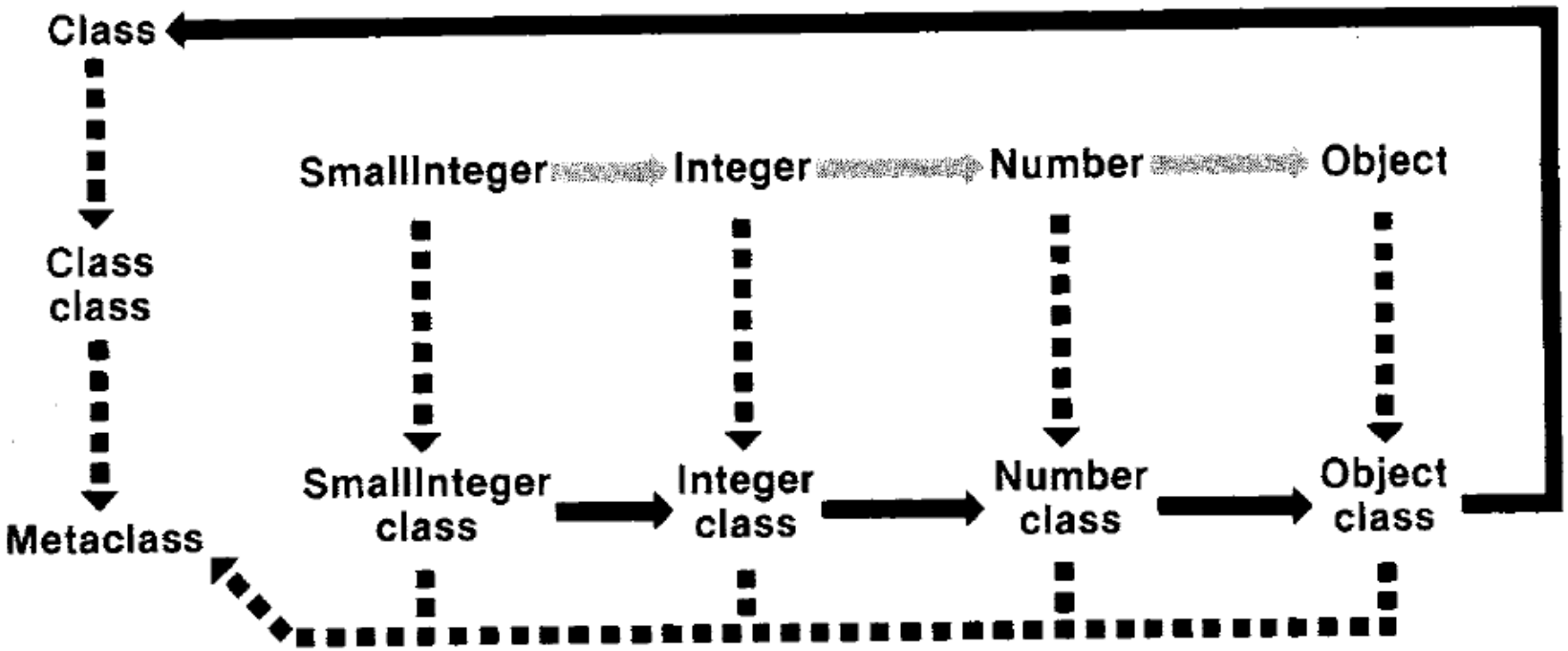
Mensajes de “clase”

Construir el programa es enviar mensajes ...

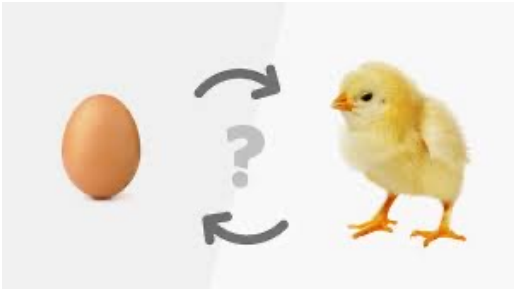
```
"Le pido a la clase un builder para crear una
subclase suya. Al builder le digo que slots
tendrá, en que paquete va, y le pido que la
construya e instale"

Object << #Persona
  slots: { #nombre. #apellido . #edad };
  package: 'OO1';
  install.

"A la clase persona le digo que compile un método
y se lo agregue en determino protocolo"
Persona compile: 'nombre ^nombre'
         classified: 'accessors'
```

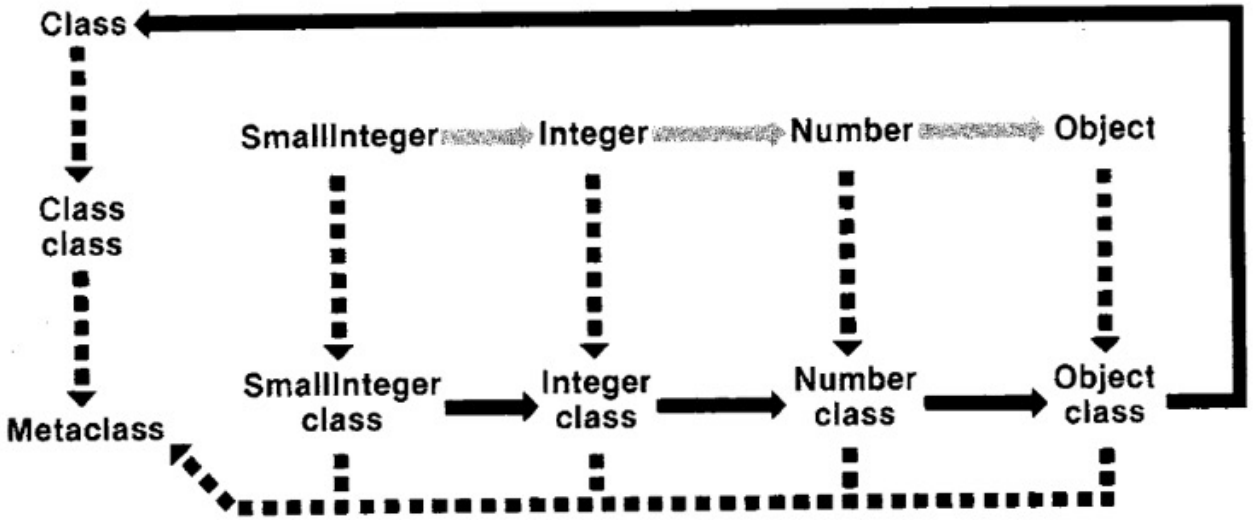


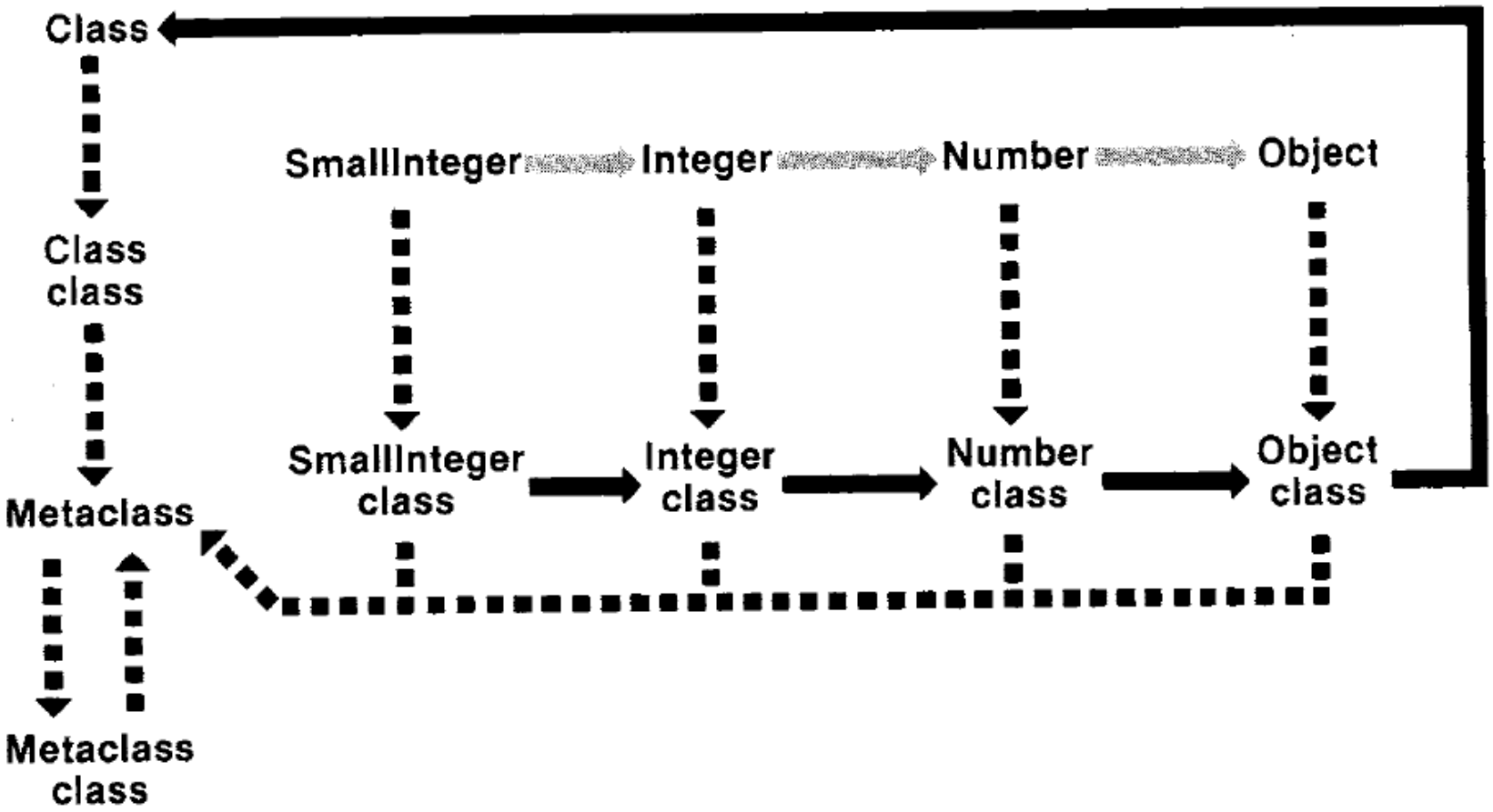

```
Integer subclass: #SmallInteger
```

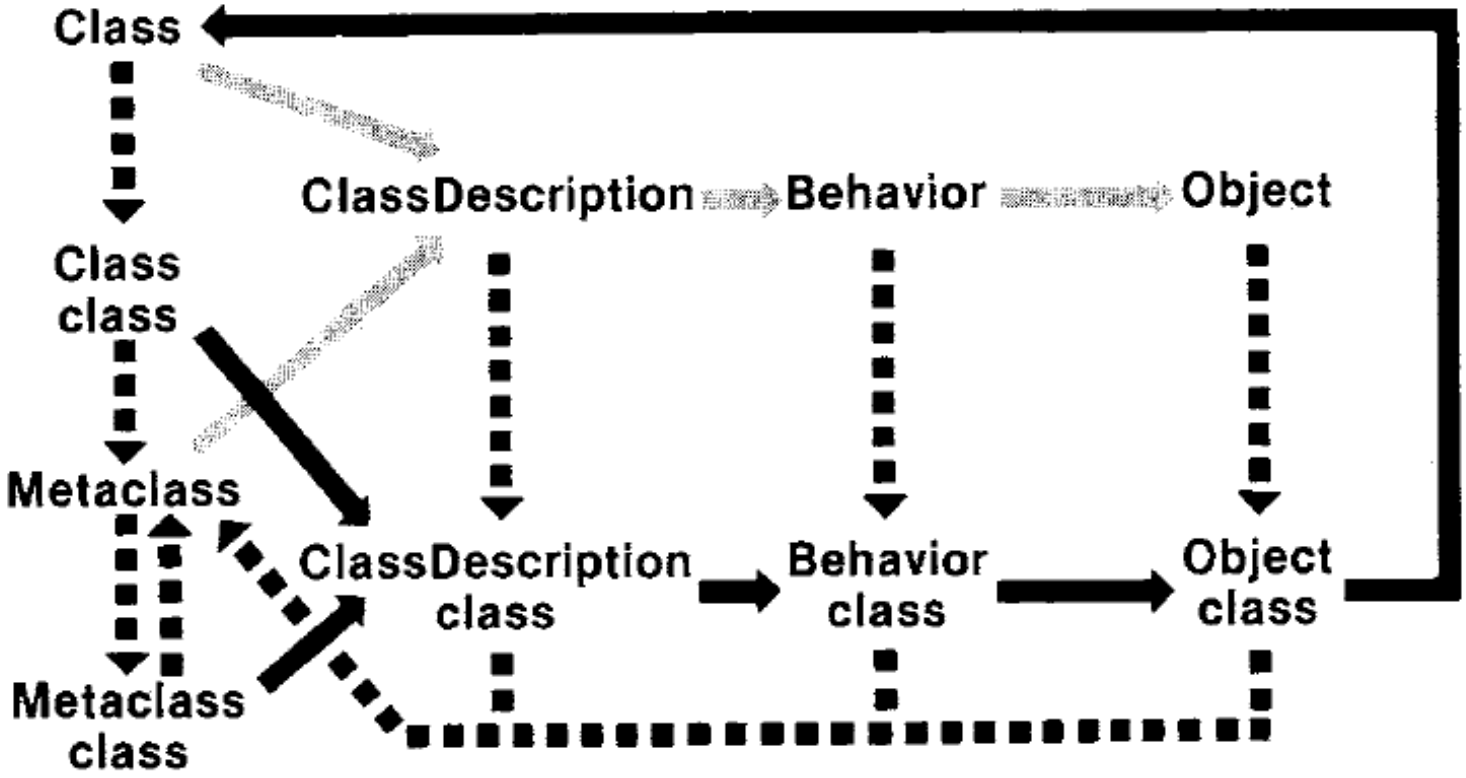

Class
SmallInteger
Integer
Number
Object
Class
class
SmallInteger
class
Integer
class
Number
class
Object
class
Metaclass

.... el huevo y la gallina en Smalltalk






Class
SmallInteger
Integer
Number
Object
Class class
SmallInteger class
Integer class
Number class
Object class
Metaclass
Metaclass class


Class
ClassDescription
Behavior
Object
Class
class
Metaclass
ClassDescription
class
Behavior
class
Object
class
Metaclass
class