


# Algunas clases de Smalltalk

Object

Magnitude



# class Object

 en Pharo, la clase Object tiene como super clase **ProtoObject**

- Todas las clases heredan de **Object**

- Object es la superclase de todas las clases,
- Todas las clases heredan sus métodos
  - provee un comportamiento común por defecto para todos los objetos
    - acceso, copia, comparación, manejo de errores, envío de errores y reflexión.
- También los mensajes útiles que todos los objetos deberían responder están definidos aquí.
- no tiene variables de instancia y no deberían agregársele.

veamos algunos de los métodos del  
protocolo de la clase **Object**



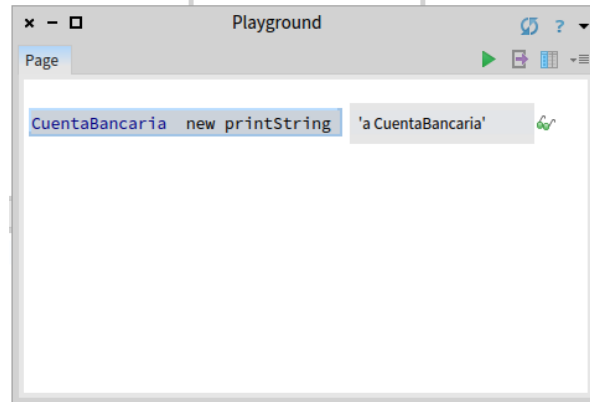
# Class Object: #printString:, #printOn:

- Todo objeto en Smalltalk puede retornar una forma impresa de si mismo.
  - miCuentaBancaria saldo -> **print It** → `'3356'`
    - print It: Opción del Menú que ejecuta la expresión y le pide al objeto retornado que se imprima
    - `'3356'` printString
- El método **printString**, el cual es un *template method*, en alguna de sus líneas de código envía el mensaje **printOn:** a su receptor.
- Object»printOn: es el método que más frecuentemente se sobreescribe.
  - La implementación por defectos escribe el nombre de la clase precedida por “a” o “an”.



# Class Object: #printString:, #printOn:

- Creamos la clase *CuentaBancaria*
  - CuentaBancaria new printString → aCuentaBancaria



- Redefinimos el método #printOn:

CuentaBancaria >> printOn: aStream

- ^ aStream
- nextPutAll: 'CuentaBancaria ';
- nextPutAll: numeroCuenta.



# Igualdad e Identidad

- el mensaje `=` comprueba la igualdad de los objetos
  - dos objetos representan el mismo valor
- el mensaje `==` comprueba la identidad del objeto
  - cuando dos expresiones representan al mismo objeto

```
| p1 p2 p3 |  
p1 := (Point x: 10 y: 20).  
p2 := (Point x: 10 y: 20).  
p3 := p2.  
Transcript open; clear; show: (p1 = p2)  
printString.  
Transcript cr; show: (p1 == p2) printString.  
Transcript cr; show: (p3 == p2) printString.
```



# Class Object (2)

## •Copiando Objetos:

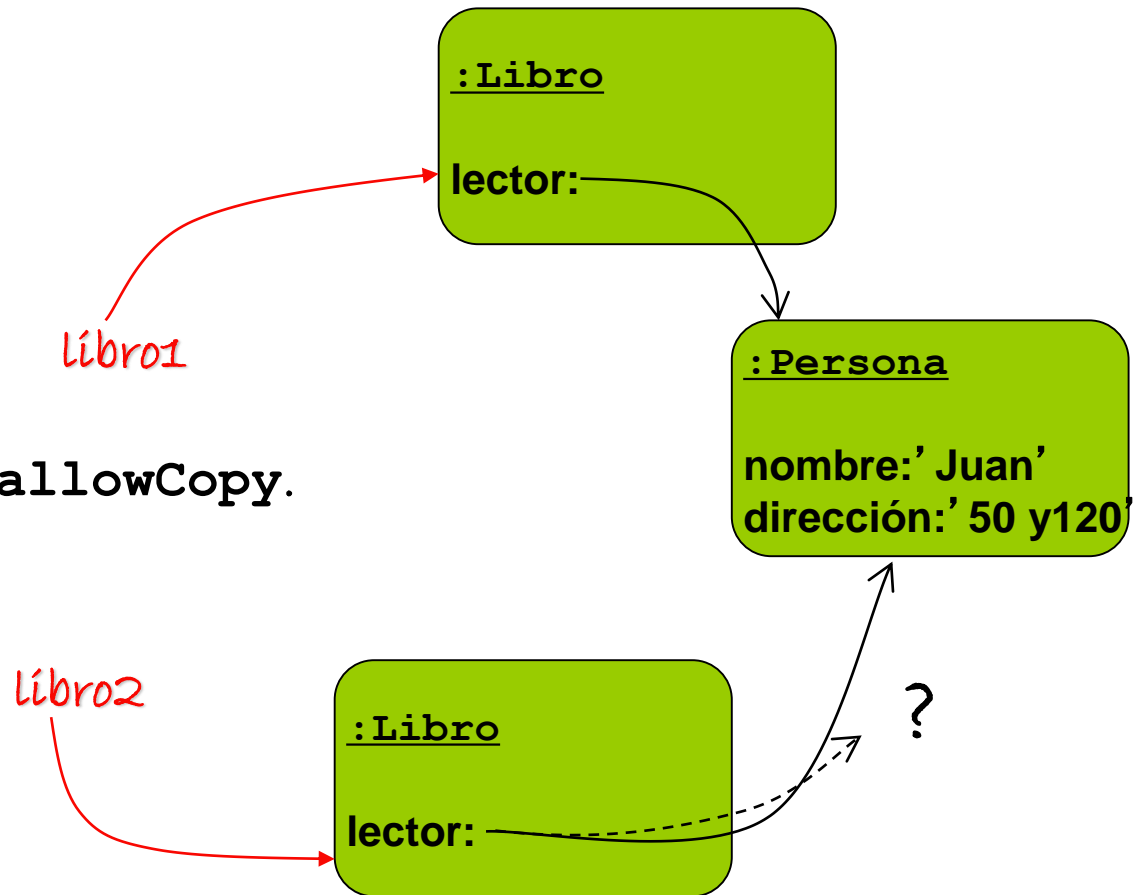
- Hace una copia del receptor y crea un nuevo objeto de la misma clase del receptor: `#shallowCopy`

```
|libro1|
```

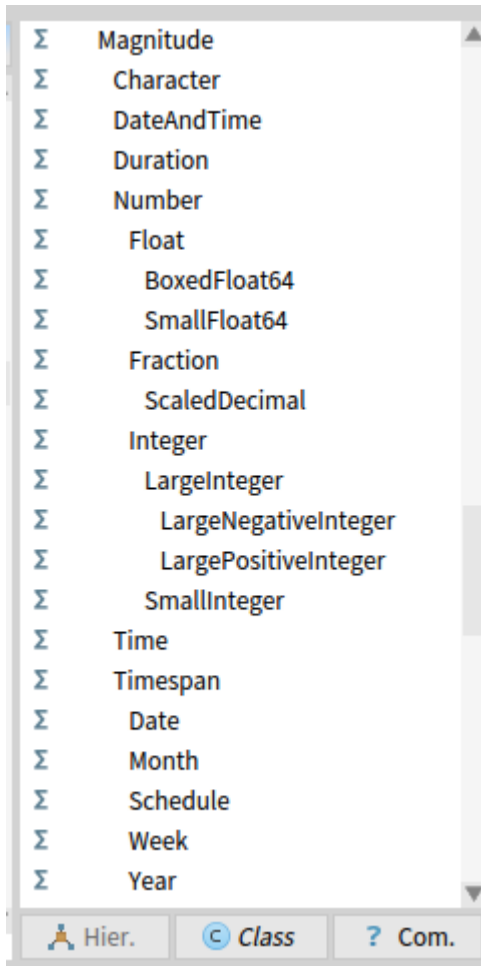
```
.....
```

```
|libro2|
```

```
libro2:= libro1 shallowCopy.
```



# Magnitude



- Es la raíz *abstracta* de las clases que soportan operadores de comparación
- Es subclase de Object

```
3 + 7 / 3  
(3 + 7 / 3) asFloat  
(3+7/3) asFixedPoint: 2  
Float pi asRational  
15 log  
0.3 sin  
1000 factorial  
37 raisedTo: 22
```

- también es superclase de Date, Time, Character, entre otras



# #between:and: y #max:

25 between:3 and:38

\$g between:\$h and:\$m

Date today between: yourBirthdays and: myBirthday

```
Magnitude>>between: min and: max
```

```
"Answer whether the receiver is less than or equal to the  
argument, max, and greater than or equal to the argument, min."
```

```
^self >= min and: [self <= max]
```

```
Magnitude>>max: aMagnitude
```

```
"Answer the receiver or the argument, whichever has the greater magnitude"
```

```
self > aMagnitude ifTrue:[^self] ifFalse:[^aMagnitude]
```





# Classes String, Character,

- String

- Colección indexada de caracteres
- `'abc' < 'xyz'`
- `'abcdefg' findString: 'de' startingAt: 1`
- `'abcdefg' size`

- Character

- `$3`
- `$a`
- `80 asCharacter` “ASCII code”
- `$a<$d`



# Creación e inicialización de objetos



# Métodos de instancia y métodos de clase

- En ST hay 2 tipos

- Métodos de instancias

- Son los que se pueden enviar a las instancias de una clase

- `'abc'.asUppercase`
      - `3/5.numerator`
      - `miCuentaBancaria.saldo`

- Métodos de clase

- Son los que se pueden enviar a las clases

- `CuentaBancaria.new`
      - `Fraction.numerator:3 denominator:5`



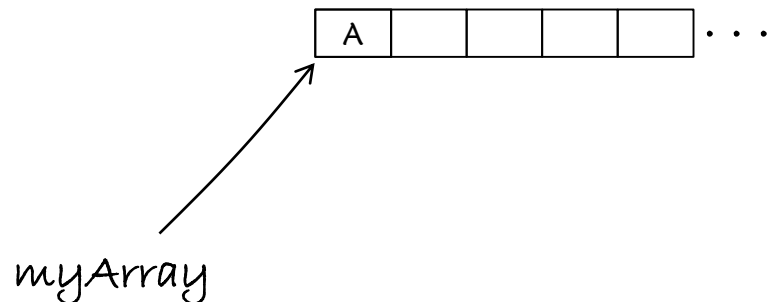
# Creación

- Se hace a través de enviarle un mensaje a una clase,
- este mensaje “generalmente” es el **#new**
  - El **#new** es un mensaje que entiende cualquier clase Smalltalk
    - Ya esta definido

**ClassName new**

una instancia de la clase **ClassName**

```
|myArray|  
myArray:= Array new:10. “crea unArray”  
myarray at:1 put: ‘A’.
```

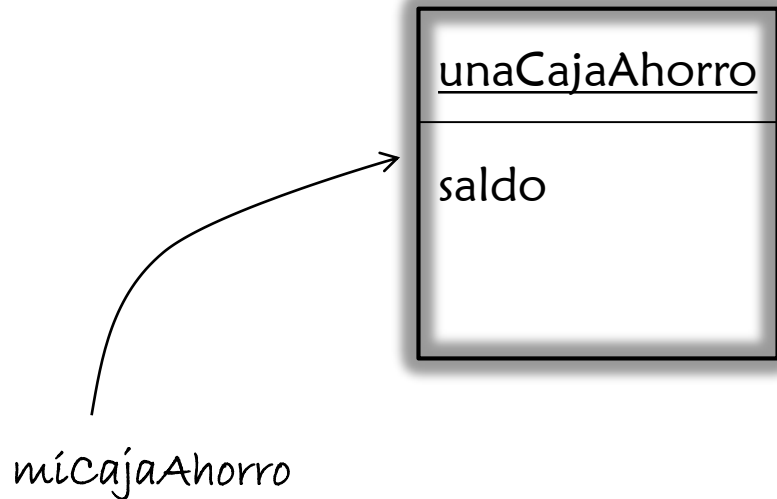


# Creación: otro ejemplo

```
|miCajaAhorro|
```

```
miCajaAhorro := CajaAhorro new. “Crea unaCajaAhorro”
```

```
miCajaAhorro depositar: 100
```



**Atención!!!**  
Smalltalk no  
maneja valores  
por defecto para  
sus variables



# Inicialización: 1ª opción

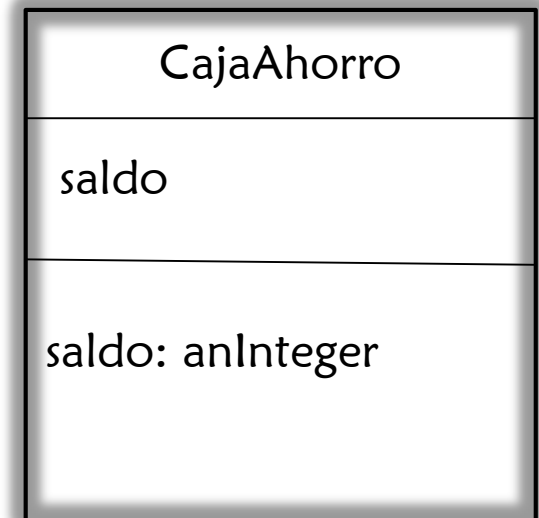
- Cuando creamos **miCajaAhorro** su **saldo** debe ser 0.

```
|miCajaAhorro|  
miCajaAhorro := CajaAhorro new saldo:0.
```

```
CajaAhorro>>saldo:unMonto
```

```
"Setea el saldo del receptor en unMonto"
```

```
saldo:= unMonto.
```



# Inicialización: 2ª opción

## Implementando el método `#initialize`

En Pharo, el método `#new` por defecto enviará `#inicialíze` a cada instancia nueva creada.

```
CajaAhorro>>initialize  
"Setea el saldo del receptor en 0"  
  
self saldo: 0.
```

```
|miCajaAhorro|  
miCajaAhorro := CajaAhorro new.  
miCajaAhorro depositar: 100
```

CajaAhorro
saldo
<i>initialize</i> <i>saldo:</i>

**`#initialize`** sirve para inicializar valores por defecto

# Inicialización: 3ª opción

- Cuando se crea una caja de ahorro se crea con un saldo  $>$  que 0, depende del valor con que la crea el cliente.

|miCajaAhorro|

miCajaAhorro := CajaAhorro new:100

```
CajaAhorro class>>new:unMonto
```

```
    "Crea unaCajaAhorro y setea su saldo en unMonto."  
    ^self new saldo: unMonto.
```

```
CajaAhorro>>saldo:unMonto
```

```
    "Setea el saldo del receptor en unMonto."  
  
    saldo:= unMonto
```

