



Introducción a Smalltalk

Alicia Díaz

`alicia.diaz@lifa.info.unlp.edu.ar`



Objetos y Clases

Objetos y Clases

- Todo objeto es una instancia de una clase

Una clase permite que todos los objetos del mismo tipo compartan la misma definición

Por ejemplo, **3/5** y **4/7** son instancias de la clase **Fraction**

La clase define las propiedades y funcionalidades (comportamiento) de sus instancias

Estado interno: **numerator, denominator**

Comportamiento: **numerator, denominator, negative, isZero**

veamos la clase Fraction y Point ...



Métodos

Métodos

- ¿Qué es un método?
 - Es la contraparte funcional del mensaje.
 - Expresa la forma de llevar a cabo la semántica propia de un mensaje particular (el cómo).
- son identificados por su clase y selector de mensaje
- El código de un método puede realizar básicamente 3 cosas:
 - Modificar el estado interno del objeto.
 - Colaborar con otros objetos (enviándoles mensajes).
 - Retornar y terminar.

Anatomía de un método

- Tiene:
 - La firma: selector del mensaje con un nombre de variable por cada palabra clave en el selector
 - Un comentario
 - Una declaración de variables temporales (entre |...|)
 - Una secuencia de sentencias
 - Si el método devuelve un objeto, se usa el ^ como carácter de retorno seguido del objeto.

Pharo! (F:\2016\Pharo5.0\Pharo5.0.image)

Pharo

Scoped Variables

Type: Pkg1|^Pkg2|Pk.*Cor...
Kernel
BasicObjects
Chronology
Classes
Copying
Exceptions
Messaging
Methods
Models
Numbers
Objects
Pragmas
Processes
Protocols
Kernel-Rules
Kernel-Tests
Keymapping-Core

ProtoObject
Object
Point

History Navigator
-- all --
accessing
arithmetic
comparing
converting
copying
extent functions
geometry
interpolating
point functions
polar coordinates
printing
private
rectangle creation
self evaluating
testing
isZero
leftRotated
max
max:
min
min:
min:max:
nearestPointAlongLineFrom:to:
nearestPointOnLineFrom:to:
negated
normal
normalized
octantOf:
onLineFrom:to:
onLineFrom:to:within:
printOn:

Hier. Class Com.

nearestPointAlongLineFrom: p1 to: p2
>Note this will give points beyond the endpoints."


```

| x21 y21 t x1 y1 |
p1 x = p2 x ifTrue: [ ^ p1 x @ y ].
p1 y = p2 y ifTrue: [ ^ x @ p1 y ].
x1 := p1 x asFloat.
y1 := p1 y asFloat.
x21 := p2 x asFloat - x1.
y21 := p2 y asFloat - y1.
t := ((y asFloat - y1) / x21 + ((x asFloat - x1) / y21)) / (x21 / y21 + (y21 / x21)).
^ (x1 + (t * x21)) @ (y1 + (t * y21))

```

Comentario

Declaración de variables

Secuencia de Sentencias

1/13 [1] Format as you read W +L

2 tipos de Métodos en ST

- 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`
 - `Date today`



Variables

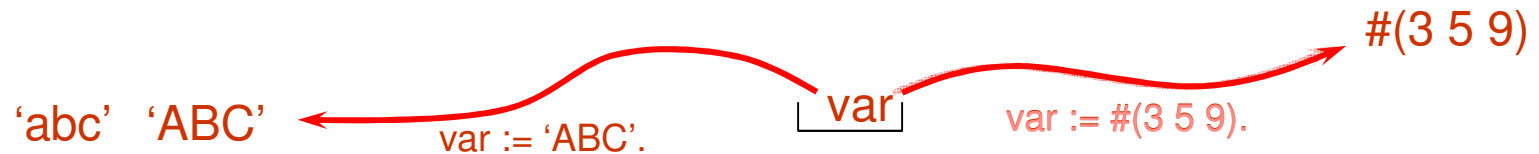
variables

- Una variable se liga a un objeto a través del operador de asignación **:=**

var := #(3 5 9).
var size

- no necesitan ser tipadas porque son punteros a objetos
- Las variables no son objetos
- Pueden ser ligadas a distintos objetos

var := 'ABC'.
var asLowercase



variables

- El nombre de una variable es una secuencia de caracteres y dígitos comenzando con una letra:
 - price
 - taxRate07
 - empleadosContratados
- Se acostumbra usar la notación *Camel* para construir el nombre de la variable
 - consiste en escribir los identificadores con la primera letra de cada palabra en mayúsculas y el resto en minúscula:
 - endOfFile.

Categoría de variables

- Variables temporales
- Variables de instancia
- Pseudo-variables: `self`, `super`, `nil`, `true`, `false`
- Variables de clases o compartidas

... veamos cada categoría

Variables Temporales

- Sirven para guarda el resultado de una ejecución que será utilizado más tarde
- deben ser declaradas antes que aparezca la sentencia que las usa
 - `|variableName|` o `|variableName1 variableName2 variableName3|`
- Se usan en el workspace o como variables auxiliares en un método
- Su alcance esta limitado al método o fragmento de código que las contenga
- Comienzan con minúscula
 - En el workspace

```
| today myBirthday daysToMyBirthday |  
  
today:= Date today.  
myBirthday := Date year: 2016 month: 12 day:13.  
daysToMyBirthday := myBirthday - today.  
^ daysToMyBirthday asDays
```

Veamos como funcionan ...

Variables Temporales

- En un método

Point>>nearestPointAlongLineFrom: **point1** to: **point2**

"Note this will give points beyond the endpoints.
Streamlined by Gerardo Richarte 11/3/97"

| x21 y21 t x1 y1 |



```
p1 x = p2 x ifTrue: [ ^ p1 x @ y ].  
p1 y = p2 y ifTrue: [ ^ x @ p1 y ].  
x1 := p1 x asFloat.  
y1 := p1 y asFloat.  
x21 := p2 x asFloat - x1.  
y21 := p2 y asFloat - y1.  
t := ((y asFloat - y1) / x21 + ((x asFloat - x1) / y21)) /  
      (x21 / y21 + (y21 / x21)).  
^ (x1 + (t * x21)) @ (y1 + (t * y21))
```

variables de instancias

- Sirven para describir las propiedades o estado interno de un objeto
- Son declaradas cuando se define una clase y las poseen cada objeto de esa clase
- Son creadas cuando se crea una instancia de una clase
- Sus valores pueden ser modificados durante la vida útil del objeto
 - Por manipulación directa *en un método de la clase a la que pertenece el objeto que las posee*
 - a través de métodos setters
- Su alcance esta limitado a los *método de instancia* de los objetos que las poseen.

variables de instancias

- Son declaradas cuando se define una clase y las poseen cada objeto de esa clase

Object subclass: **#Point**

```
instanceVariableNames: 'x y'  
classVariableNames: ''  
category: 'Kernel-BasicObjects'
```

Number subclass: **#Fraction**

```
instanceVariableNames: 'numerator denominator'  
classVariableNames: ''  
category: 'Kernel-Numbers'
```

OTFRobot subclass: **#WalkingBrushRobot**

```
instanceVariableNames: 'battery state'  
classVariableNames: ''  
category: 'BotArena-Robots'
```

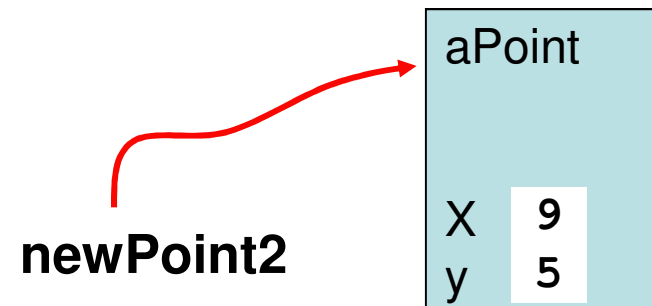
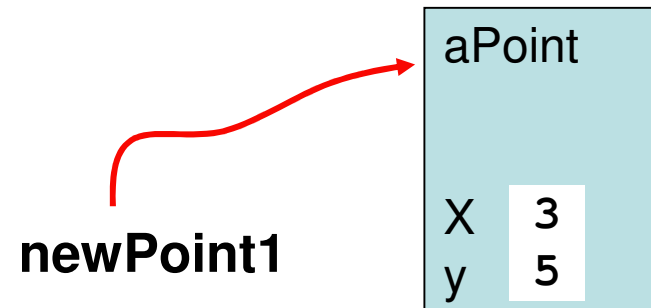

variables de instancias

- Son creadas cuando se crea una instancia de una clase

```
| newPoint1 newPoint2|  
newPoint1 := Point new.  
newPoint2 := Point new.
```

```
newPoint1 x: 3.  
newPoint1 y: 5.
```

```
newPoint2 x: 9.  
newPoint2 y: 5.
```



variables de instancias

- Sus valores pueden ser modificados durante la vida útil del objeto
- Por manipulación directa *en un método de la clase a la que pertenece el objeto que las posee*

```
Point>>x: xInteger  
      "Set the x coordinate."  
      x := xInteger
```

Variable de instancia definida en Point

- a través de métodos setters

```
| newPoint |  
  newPoint := Point new.  
  newPoint x: 3.  
  newPoint y: 5.
```

Setters de Point

Pseudo Variables

- Variables que son asignadas a algún objeto por el compilador y no pueden ser modificadas a través del :=
- Su binding existe solamente durante la ejecución del método, sin embargo el objeto puede continuar existiendo
- El alcance de la variable es el método que las contiene
- Hay 3 tipos:
 - Argumento de los mensajes

`CuentaBancaria>> extraer: unMonto`

`- nil, true, false`

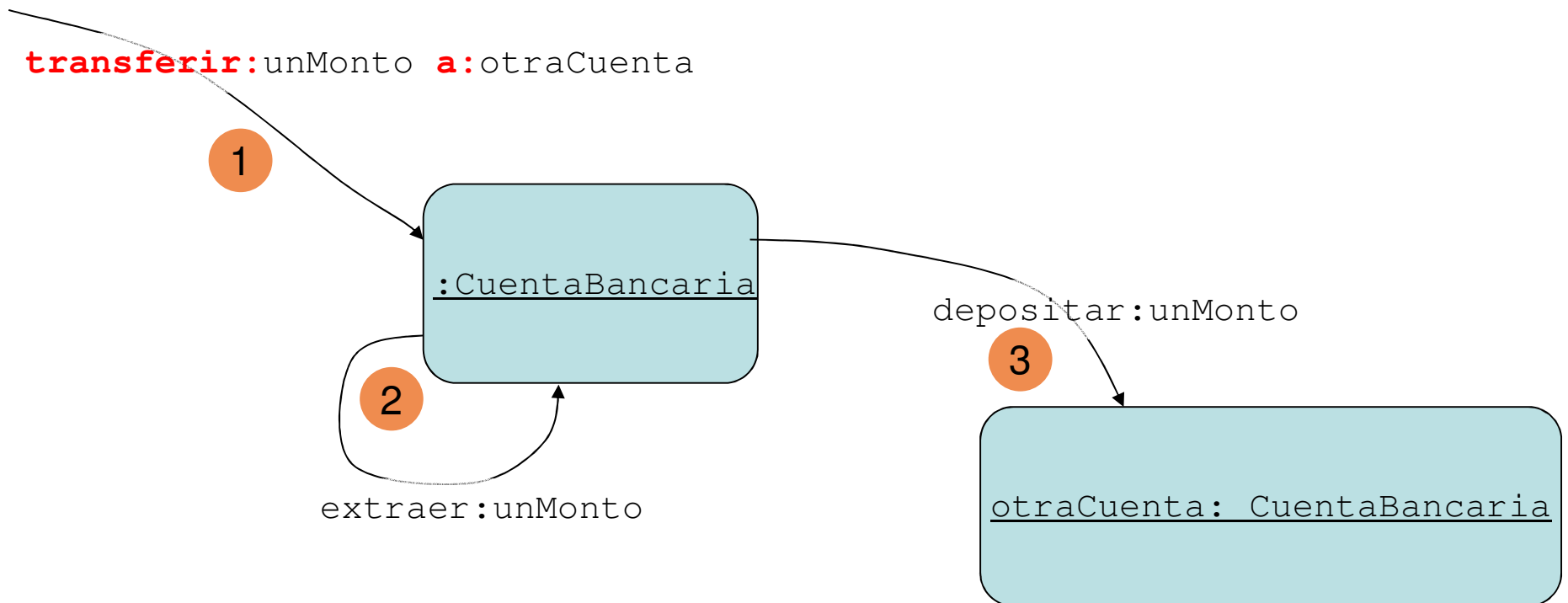
`.....
^false
... ..`

`- self, super`

Pseudo variables: **self**

- Una cuenta bancaria sabe “transferir un monto desde ella a otra cuenta bancaria”

CuentaBancaria>>transferir:unMonto a:otraCuenta



CuentaBancaria>>transferir:unMonto a:otraCuenta
self extraer:unMonto.
otraCuenta depositar:unMonto.

Otro ejemplo

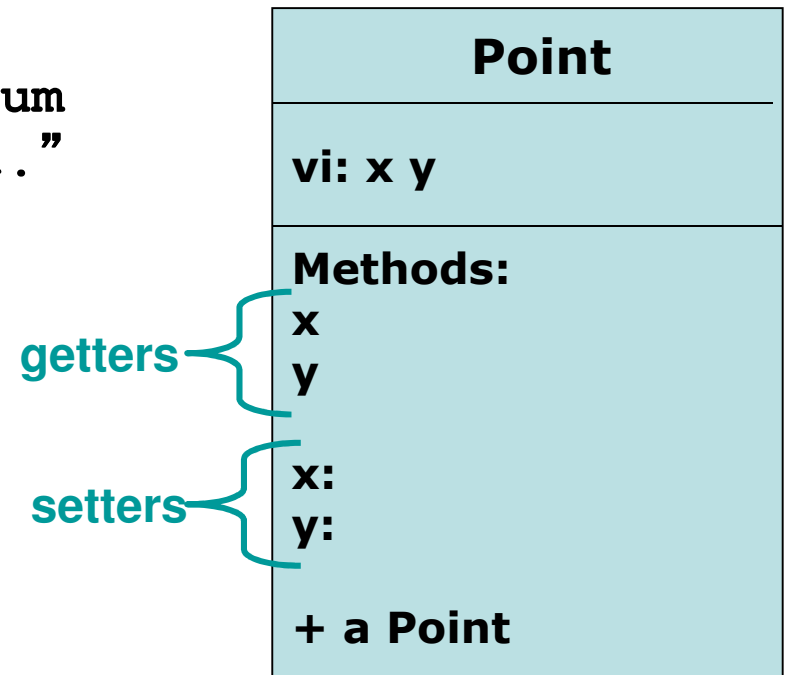
- Método **+** de la clase Point

– 3@5 + (4@9)

+ aPoint

“Answer a new point that is the sum of the receiver and the argument.”

```
|newPoint|  
newPoint := Point new.  
newPoint x: self x + aPoint x.  
newPoint y: self y + aPoint y.  
^ newPoint
```



condicionales e iteradores

- Algunos Condicionales

```
aBoolean ifTrue:[ sentencias... ]
```

```
a>b iftrue:[^"a es mayor que b"]
```

```
aBoolean    ifTrue:[ sentenciasTrue... ]  
            ifFalse:[ sentenciasFalse... ]
```

```
a>b    ifTrue:[^"a es mayor que b"]  
       ifFalse:[^"b es mayor o igual que a"]
```

- Otros

```
ifFalse: ; ifFalse: ifTrue:
```

condicionales e iteradores

- Algunos Iteradores

- timesRepeat:

```
anInteger timesRepeat: [ sentencias... ]  
    | sum |  
    sum:=0.  
    4 timesRepeat:[sum:= sum+10]
```

- to:do:

```
anInteger to:otherInteger do:[:index| sentencias(index)]  
    | sum |  
    sum:=0.  
    1 to:10 do:[:i|sum:= sum+i]
```

- to:do:by:

¿En qué clase están definidos estos mensajes? ...