Universidad Tecnológica Nacional
FACULTAD REGIONAL CORDOBA

PARADIGMAS DE PROGRAMACION

Unidad III

Paradigma de Programación con Orientación a Objetos

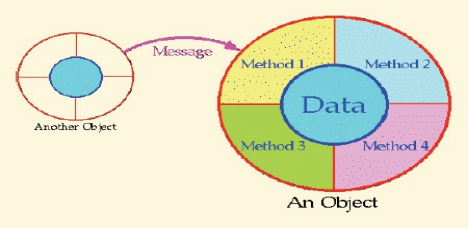
Lenguaje Smalltalk – Parte 2

CONTENIDOS ABORDADOS

- Lenguaje Smalltalk:
  - Bloques.
  - Abstracciones de control.

Repaso Smalltalk

- Smalltalk es un **lenguaje orientado a objetos puro**, todas las entidades que maneja son objetos.
- La programación consiste en:
  - **Crear clases.**
  - **Crear instancias.**
  - **Especificar la secuencia de mensajes entre objetos.**



Bloques en Smalltalk

- Los bloques son objetos que representan secuencia de expresiones Smalltalk (código).
- Un bloque está formado por una secuencia de expresiones separadas por punto y encerradas entre corchetes.

```
[ expresion1.
  expresion2.
expresion3. ]
```

- El principal uso de los bloques es en la construcción de estructuras de control. Además, pueden utilizarse para construir objetos que encapsulen algoritmos.
- El bloque se ejecuta cuando recibe el mensaje **value**.

Bloques en Smalltalk

- Las siguientes reglas se aplican a los bloques:
  - Un bloque puede contener cualquier número de expresiones válidas, o cualquier número de comentarios.
  - Cada expresión debe terminar con un punto, excepto cuando se definen las variables temporales, y en la última expresión del bloque, donde el punto es opcional.
  - Un bloque tiene acceso a las mismas variables que el método al que pertenece. Como un bloque es parte de un método no posee una definición de interface de método.

Bloques en Smalltalk

Ejemplos:

```
| a b c |
[ a := 5. b := 7. c := a  + b  ].
```

- La secuencia de expresiones contenida en el bloque se ejecuta cuando el bloque recibe el mensaje **value**.
- El resultado de la evaluación del bloque es igual al valor de la última expresión del bloque.
- Si evaluamos el bloque del ejemplo anterior, obtenemos:

```
| a b c |
[a:= 5.      b:= 7.  c:= a + b  ] value
```

Resultado: 12

Bloques en Smalltalk

- Los bloques constituyen objetos como cualquier otro, por lo cual pueden ser asignados a variables:

```
| a b c  unBloque|
unBloque := [ a:= 5. b:= 7.  c:= a + b ].
unBloque value.
```

Resultado: 12

- Un bloque sin expresiones se denomina bloque nulo y al ser evaluado devuelve nil.

```
| bloqueNulo |
bloqueNulo :=  [ ].
bloqueNulo  value.
```

Resultado: nil

Bloques en Smalltalk

- Un bloque puede recibir ninguno, uno o dos parámetros. En cada uno de esos casos la sintaxis es la siguiente:

  [… expresiones …]. “ningún parámetro”

  [: x | … expresiones …]. “un parámetro”

  [:x :y | … expresiones …]. “dos parámetros”

- Los mensajes para evaluar bloques de uno y dos parámetros son, respectivamente:

  bloque value: parámetro

  bloque value: parámetro1 value: parámetro2

Bloques en Smalltalk

"Ejemplo de un bloque de un parámetro"

```
| unBloque res1 res2|
unBloque := [ :x |  x*2 ].
```

Paso de mensajes al bloque con un parámetro:

```
res1 := unBloque value: 2.        Resultado: 4
res2 := unBloque value: 3.        Resultado: 6
```

"Ejemplo de un bloque de dos parámetros"

```
| unBloque res1 res2|
unBloque := [ :x  :y |  x + y ].
```

Paso de mensajes al bloque con dos parámetros:

```
res1 := unBloque value: 2 value: 3.  Resultado:5
res2 := unBloque value: 5 value: -8. Resultado:-3
```

Control en Smalltalk

Abstracciones de control

- Las estructuras de control se encapsulan en abstracciones de control, que se construyen a partir de mensajes enviados a objetos booleanos, bloques y números, y de la cooperación entre ellos.

Lógica condicional

- La lógica condicional permite la ejecución del código dependiendo de un valor booleano.

```
Boolean mensaje: bloque.
```

Control en Smalltalk

Lógica condicional

```
Boolean mensaje: bloque.
```

Ejemplos:

(a>b) ifTrue: [c:= a + b]. "Ejecuta el bloque si la condición es verdadera"

(a=b) ifFalse: [c:= a + b]. "Ejecuta el bloque si la condición es falsa."

(a>=b)
ifTrue: [c:= a + b]
ifFalse: [c:= a - b].

"Ejecuta el bloque [c:= a + b] si la condición es verdadera; sino ejecuta el otro bloque."

Control en Smalltalk

Iteraciones

- Smalltalk soporta cuatro tipos tradicionales de iteraciones
- Ellos son:
  - Hacer algo n número de veces: timesRepeat.
  - Hacer algo usando un índice, comenzando con un valor inicial, y finalizando en un valor final: to:do:
  - Hacer algo mientras se encuentre con una condición True: whileTrue.
  - Hacer algo mientras se encuentre con una condición False: whileFalse.

Control en Smalltalk

timesRepeat

- El mensaje timesRepeat: ejecuta un bloque de código un número específico de veces. El formato del mensaje es:

  número timesRepeat: [código]

  donde número puede ser cualquier expresión que resulte en un entero, y código es un bloque de código (expresiones) de cero-argumento. Ejemplo:

  "Agrega 1 a la variable x tres veces.“

```
| x |
x := 2.
3 timesRepeat: [x := x + 1].
Transcript show: x.
```

El resultado es 5

Control en Smalltalk

to: do:

- El mensaje to:do: ejecuta un bloque múltiples veces, basado en un valor inicial y un valor final. El formato del mensaje es:

  número1 to: número2 do: [:var | código].

  Ejemplo:

  "Ejecuta este bloque 3 veces con i referenciado a cada valor entre el rango de 1 a 3. “

```
| x |
x := 0.
 1 to: 3 do: [:i | x := x + i].
Transcript show: x.
```

El resultado es 6.

Control en Smalltalk

to: do:

"Itera de 1 hasta 10, imprimiendo los sucesivos números en la ventana Transcript."

```
1 to: 10
do: [ :i | Transcript show: i; cr].
```

"Itera de 0 a -10, de a dos valores por vez, e imprime esos valores en la ventana Transcript."

```
0 to: -10
by: -2
do: [ :i | Transcript show:  i; cr.].
```

Control en Smalltalk

whileTrue: y whileFalse:

- Estos dos mensajes realizan la misma operación, excepto que uno se ejecuta por true y el otro por false.
- El formato del mensaje es:

  [booleano] whileTrue: [código].

  [booleano] whileFalse: [código].
- Un booleano puede ser cualquier expresión que resulte en un valor de true o false; debe estar encerrado en un bloque. La expresión [código] es un bloque de código de cero-argumento.

Control en Smalltalk

whileTrue: y whileFalse:

"Itera mientras y es menor o igual que x“

```
| x y |
x := 5.
y := 0.

[y <= x] whileTrue:
   [y := y + 1].

Transcript show: y.
```

"Itera mientras x es mayor que y“

```
| x y |
x := 5.
y := 0.

[x < y] whileFalse:
   [y := y + 1].

Transcript show: y.
```

Control en Smalltalk

whileTrue:

"Muestra en la ventana Transcript cada total obtenido en el calculo del factorial de un número k"

```
|k total|
k:= 1.
total:= 0.
[k<10]
whileTrue: [
    total:= total + k factorial.
    Transcript show: total; cr.
    k := k + 1 ].
```

"Repite el bloque mientras la condición k<10 sea verdadera."

Control en Smalltalk

whileFalse:

"Muestra en la ventana Transcript el factorial de un número k"

```
|k total|
 k:= 10.
 total:= 0.
 [k=0]
 whileFalse: [
   total:= total + k factorial.
   k := k - 1 ].

Transcript show: total; cr.
```

"Repite el bloque mientras la condición **k=0** sea falsa."