Universidad Tecnológica Nacional – Facultad Regional Buenos Aires

Cátedra de Paradigmas de Programación

Sintaxis Smalltalk

Autores:

- Nicolás Pérez Santoro – nicolas.perez.santoro@gmail.com
- Carlos Lombardi – carlombardi@gmail.com

Historial de Revisiones

Versión	Revisado por	Fecha	Detalles
V 1.0	Carlos	02/09/2008	Versión Inicial, se edita.
V 1.1	Carlos	16/09/2008	Métodos que esperan vs. parámetros

Índice

Sintaxis, mensajes y métodos 2
Mensajes 2
Mensajes unarios 2
Mensajes binarios 2
Mensajes de palabras claves (keyword messages) 2
Cosas que no son mensajes 3
Precedencia 3
Sintaxis de un método 4
Ejemplo 4
Distinta cantidad de parámetros 5

Sintaxis, mensajes y métodos

Mensajes

En smalltalk hay tres tipos de mensajes, las diferencias son puramente sintácticas.

Mensajes unarios

Sintaxis: objeto mensaje
Ejemplo:

```
9 squared
```

(9 elevado al cuadrado)
donde "9" es el objeto al cual le envío el mensaje, y "squared" es el nombre del mensaje. Los mensajes unarios no tienen parametros

Mensajes binarios

Sintaxis: objeto mensaje parametro
Ejemplo:

```
2 + 3
```

Los mensajes binarios tienen como nombre del mensaje simbolos como "+", "-", "<", lo que en otro lenguaje serían primitivas de operaciones aritméticas, en Smalltalk son mensajes (que se pueden definir para cualquier objeto, como cualquier otro mensaje). Para sumar 2 + 3, se envía el mensaje "+" al número 2, con el objeto 3 como parametro del mensaje. Los mensajes binarios siempre tienen exactamente un parametro.
Los mensajes binarios pueden ser usados para cosas distintas de la aritmética, por ejemplo si a un String le envío el mensaje , con otro String como parámetro, me devuelve la concatenación. Y si le envío a un número el mensaje @ con otro número como parámetro, devuelve un punto de dos dimensiones con el receptor y el parámetro como coordenadas, p.ej.

```
2@5
```

Mensajes de palabras claves (keyword messages)

Sintaxis: objeto palabra1: parametro1 palabra2: parametro2 palabra3: parametro3 ...
Ejemplos:

```
2 raisedTo: 4
```

(2 elevado a la 4ta, es decir 16)

```
'hellohorum' copyReplaceAll: 'h' with: 'c'
```

(reemplaza las ocurrencias de 'h' en la palabra 'hello' con 'c', la string que devuelve el mensaje es 'cellocorum')

Los mensajes de palabras claves tienen uno o más parametros. El "nombre" del mensaje es el de todas las palabras claves, en el primer ejemplo tenemos el mensaje "raisedTo:", en el segundo tenemos el mensaje "copyReplaceAll:with:". Hay una palabra clave por cada parametro que se quiere enviar.

Cosas que no son mensajes

Las únicas cosas que pueden aparecer en el código y que no son mensajes son:

- la indicación de que un método termine devolviendo un objeto, que se indica con el circunflejo (^).
- la asignación, o sea, indicarle a una variable que haga referencia a un objeto determinado; esto se indica con :=.

Precedencia

En Smalltalk, los mensajes se evaluan con la siguiente precedencia:

a. Primero los mensajes unarios.
b. Luego los mensajes binarios.
c. Después los mensajes de palabras claves.
d. Después de todos los mensajes, la asignación.
e. Por último el circunflejo.

Y dentro de la misma precedencia, de izquierda a derecha.

Si queremos forzar otro orden de evaluación, debemos utilizar parentesis.

Ejemplo 1

```
2 squared + 3 squared raisedTo: 2
```

En este caso, se evalúan en este orden:

```
((2 squared) + (3 squared)) raisedTo: 2
```

a. los mensajes "squared" (mensajes unarios), "2 squared" devolviendo el objeto 4, "3 squared" devolviendo el objeto 9.
b. el mensaje "+" (mensaje binario), que se envía al objeto 4 con el objeto 9 como parametro. El mensaje "+" devuelve el objeto 13.
c. el mensaje raisedTo: (mensaje de palabra clave), que se envia al objeto 13 con el objeto 2 como parametro.

Ejemplo 2

```
3 + 4 * 2
```

En este caso tenemos solamente mensajes binarios. Esto no da el resultado 11, es decir 3 + 8, sino que da 14: Smalltalk evalua los mensajes binarios, como ya se dijo, de izquierda a derecha: primero suma (3 + 4), luego multiplica (7 * 2). Esto es porque "+" y "*" son solo mensajes comunes y corrientes que se envían a objetos, que en este caso representan números, smalltalk no conoce las reglas de la precedencia de las matemáticas. Esto deberia haberse escrito "(3 + 4) * 2" o "3 + (4 * 2)" según lo que se quiera hacer (generalmente se utilizan parentesis por claridad incluso si lo que se quiso escribir fue "(3 + 4) * 2".

Ejemplo 3

```
2 raisedTo: 2 raisedTo: 2
```

Podríamos pensar que esto devuelve 2 elevado al cuadrado dos veces seguidas, pero no: smalltalk entiende que estamos enviando un único mensaje "raisedTo:raisedTo:" con dos palabras claves (este mensaje no existe), es decir lo interpreta como el mensaje "copyReplaceAll:with:" que vimos antes.
Entonces, debería haberse escrito así:

```
(2 raisedTo: 2) raisedTo: 2
```

Ejemplo 4

```
valorNuevo := valorViejo raisedTo: 2 + 3.
```

En este caso, valorNuevo va a hacer referencia al resultado de elevar a la quinta al valorViejo, porque la asignación tiene la precedencia más baja.

Sintaxis de un método

En la definición de un método, debemos escribir el nombre del mensaje y el nombre de los parametros (en caso de no ser un mensaje sin parametros, es decir unario).
Luego, una linea debajo, podemos, opcionalmente, listar las variables internas del método entre "|" (pipes).
Y luego viene el cuerpo del metodo, que son un grupo de sentencias, en las cuales indicamos el valor a devolver con el simbolo "^" (todos los metodos devuelven un valor, si no se indica por defecto se devuelve el mismo objeto al que se le envió el mensaje).

Ejemplo

Supongamos que tenemos un objeto que representa a la golndrina `pepita`, con las variable "ubicacion", que hace referencia a un lugar; y "vueloMaximo" que hace referencia a un número.
De los lugares sabemos que entienden el mensaje distanciaHasta: que espera otro lugar como parámetro.

Queremos que pepita entienda el mensaje "andaA:", recibiendo otro lugar como parámetro.
Lo que queremos que haga el método es de pepita, y hacer que vuele la distancia entre donde está y el nuevo lugar, y luego reasignar su ubicación. Antes vamos a validar que la distancia no supere el vuelo máximo que puede hacer pepita.
En todos los casos, devuelve el objeto que representa al lugar donde quedó pepita.

Esta es una definición del método `andaA:`

```
andaA: otroLugar
      "me piden que cambie de lugar"
      | distancia |

      distancia := ubicacion distanciaHasta: otroLugar.
      (distancia > vueloMaximo)
            ifTrue: [self vola: distancia.
                        ubicacion := otroLugar].
      ^ubicacion
```

Analicemos las partes del mensaje.
La primer línea define el nombre del mensaje, "andaA:"; "unNumero" es la variable que representa el parametro.

Después viene un comentario que explica el efecto del método, está pensado para quien va a usar el método más que para quien lo tenga que modificar.

"|distancia|" dice que hay una variable interna del método, que se llama "distancia".

Después viene el cuerpo del método, que incluye varios mensajes y dos asignaciones, una a la variable interna, la otra a una variable del objeto que recibió el mensaje.
Finalmente, con "^ubicacion" devolvemos el objeto que representa la ubicación de pepita luego del método. El "^" es como el "return" de C y otros lenguajes, si no lo pusieramos, el método devolvería como resultado el objeto al que se le envió el mensaje, en este caso pepita.

Distinta cantidad de parámetros

Si el mensaje que corresponde al método que queremos escribir lleva más de un parámetro, armamos el nombre del mensaje en forma análoga a cómo se envía un mensaje con varios argumentos, que es

```
parte1DelNombre: arg1 parte2DelNombre: arg2 ...
```

p.ej. si quiero que pepita entienda el mensaje andaA:comiendo:, que recibe un lugar y una cantidad, el método correspondiente va a empezar así

```
andaA: otroLugar comiendo: gramos
      "me piden que cambie de lugar y de paso coma algo"
      ... variables y código ...
```

Lo mismo si no tiene parámetros, p.ej. si quiero que pepita entienda el mensaje dateUnaVuelta, el método correspondiente empieza así

```
dateUnaVuelta
      "me piden que pasee un rato por donde esté"
      ... variables y código ...
```