Universidad Tecnológica Nacional
Facultad Regional Buenos Aires
Paradigmas de Programación

Objetos Básicos de Smalltalk

Agosto de 2008
Autores: Mariana Matos y Guillermo Polito

¿Que son los objetos básicos?

Teniendo en cuenta que ya sabemos qué es un objeto y como hablarle a ese objeto (y presuponiendo que en clase les llenamos la cabeza diciéndoles “en objetos todo es un objeto”), avanzamos un pasito más, para hablar de los objetos básicos de Smalltalk. ¿Qué son? Bueno, son todos esos objetos que ya vienen con smalltalk y están a nuestro servicio para cumplir muchas operaciones básicas y otras no tan básicas.

Por ejemplo, los objetos que para todos deben ser los más básicos de todos, y que usamos sin haber tenido mucha conciencia de ello, son los números. Hay también otros como son los caracteres, las cadenas de caracteres o “strings”, las fechas, la hora, los booleanos (true y false, para el que los recuerde de pascal luego de unas largas vacaciones en C)...

La idea de este apunte es darles un pantallazo de algunas de las muchas cosas que vienen con la imagen del Smalltalk, para que puedan desenvolverse con mayor facilidad en la materia, tanto en los trabajos prácticos, como en los parciales. ¡Pero ojo! Esto es solo un esbozo de lo que van a encontrar si se ponen a investigar. Smalltalk no se queda en lo poquito que vamos a ver en estas hojas, puesto que pese a que tenga poca “popularidad” o sea “poco comercial”, es un lenguaje muy poderoso y sencillo de usar.

Los Números

Los números, como todo, son objetos. Y por ser objetos entienden mensajes. Yendo a lo concreto, algunos de los mensajes que se le pueden enviar a un número son:

- los que realizan operaciones aritméticas: +, -, /, *, //, \\

2 + 3 “Si lo pinto y lo ejecuto, retorna 5”
1.5 / 9 “Da 0.1666666666667”

16//5 “Esta es la división entera, da 3”
5\\3 “Este es el resto, da 2”

- los que los comparan con otro número: <, >, =, <=, >=

  2>3 “Este me dice que si”
  4>=5 “Este me dice que no”

- los que hacen funciones más locas:

  0 cos “El coseno de 0 es 1”
  0 sin “El seno de 0 es 0 ;). También esta la tangente que es tan”
  0 min: 15 “Este me dice el mínimo entre estos 2, en este caso retorna 0”
  15 max: 12 “Este me dice el máximo, o sea, 15”
  9 sqrt “Todos acá sabemos cuanto da la raiz de 3, ¿no?”
  3 raisedTo: 2 “¿Y 3 elevado a la 2?”
  -5121 abs “El valor absoluto de -5121...”

- los que pueden servir para otras cosas interesantes:

  even → para preguntar si es par
  odd → si es impar
  isZero → Adivinen =P
  negative → si es negativo
  negated → Negado

Todos los mensajes que hacen cuentas o aplican funciones, no modifican el objeto al que se le manda el mensaje, sino que retornan un objeto completamente nuevo.

Y muchos otros tantos que pueden investigar por su cuenta... Por el momento, y sin marearse, pueden abrir el "class browser" y buscar *Number*. Ahí van a encontrar en el listado de métodos muchas otras cosas. Y si bajan en la jerarquía a *Float*, *Integer*, *SmallInteger*, van a encontrar mensajes especiales que entienden sólo esos tipos de números más específicos.

Caracteres

Un carácter.... es un carácter. ¿Que diferencia tienen estos caracteres con los que ya conocemos? Y... empezando porque estos son objetos, que entienden mensajes y hacen cosas (los de C y pascal no servían para nada, ahora lo van a ver...)

En Smalltalk los caracteres se escriben precedidos por un signo $, y se les pueden enviar algunos mensajes como:

$a isAlphaNumeric "Las letras son alfanuméricas, sí"
$B isLowercase "Este me dice que no, porque esta en mayúsculas"
$B isUppercase "y ahora que sí"
$1 isDigit "¿El 1 es un dígito?"

$1 digitValue "Esto retorna el número 1, que es distinto del carácter 1"
$a asString "Esto me da la cadena o string 'a'. En la próxima sección vamos a hablar de eso"
$q asciiValue "El valor ascii. ¿Hace falta aclarar mas?"

"Los caracteres pueden compararse entre sí, para ordenarse alfabéticamente"
$a > $b "Nop, la b es mayor a la a, porque esta después"
$a = $A "Nop, se da cuenta que son distintos ;) "
$a > $A "Sip, porque las minúsculas vienen antes que las mayúsculas para el sr Smalltalk"

Si hablamos de caracteres, con este pequeño listado de mensajes creo nos quedamos cortos. Buscando *Character* en el *class browser* podrán encontrar otros mensajes para los caracteres.

Strings

Los *strings* son cadenas de caracteres (o texto, en criollo) y en Smalltalk se escriben entre comillas simples como 'este es un string'. Puede que sea muy repetitivo decir que son objetos y entienden mensajes, así que vamos a lo importante. ¿Qué podemos decirle a un string? Creo que ya no hace falta decir cómo....

- podemos preguntarle cosas sobre sí mismo:

  'lalalalala' size "me retorna la longitud, 10 en este caso"
  'unBarquito' beginsWith: 'unB' "me dice que sí porque empieza con 'unB'"
  'unBarquito' beginsWith: 'unb' ignoreCase: true "true, porque ignora las mayúsculas"
  'unString' includes: $r "devuelve true, pues el string contiene el carácter r".

- podemos compararlos alfabéticamente:

  'hola' > 'chau' "Lo que resulta falso :)"
  'a' >= 'aaaa' "Este es falso"
  'hola' = 'hola' "verdadero"
  'Hola' = 'hola' "falso, hace diferencia entre mayúsculas y minúsculas".

- formatearlos un toque:

  'capitalizar' capitalized "retorna 'Capitalizar', en letra capital".
  'mayusculas' asUppercase "me da 'MAYUSCULAS'"
  'MiNuScUlAs' asLowercase "me retorna 'minusculas'"
  'sotejbo' reverse "devuelve 'objetos'"

  Todos estos métodos de formateo, retornan un *string* distinto al original, nunca modifican al *string* al que le mando el mensaje.

Pueden encontrar todavía muchos más mensajes que pueden enviarle a los *strings*. ¿Dónde? Buscando *String* donde siempre, en el *class browser*.

Bloques

Lo que llamamos un bloque, es un bloque de código. Un bloque como:

[1+1]

Dentro de un bloque podemos poner el código que nosotros queramos, desde mandar mensajes (que es lo que normalmente hacemos con los objetos) hasta declarar variables y utilizarlas.

Pese a todo esto, un bloque no deja de ser un objeto. ¿Y dónde se ve eso?
Principalmente al saber que un bloque no se ejecuta hasta que alguien le mande el mensaje *value*. Por ejemplo:

bloque := [1+2]. "si inspeccionamos luego de ejecutar esta línea, vamos a ver que es un bloque, no el 3".
bloque value. "si ejecutamos, el mensaje value se envia al bloque, y nos retorna 3."

¿Por qué que bloque retornó 3 si no le puse el circunflejo? Porque el bloque retorna la última expresión que se ejecutó dentro de él. Qué pasa si pongo un circunflejo (^) en el bloque, más adelante.

Un bloque puede conocer a las variables de afuera de él, como por ejemplo, el método vola de pepita:

```
#Pepita
>vola: unosKilometros
  | energiaConDesgaste |
  energiaConDesgaste := energia – 5.
  [ self descontarEnergia: energiaConDesgaste conInicial: energia ] value.
```

Independientemente de la incoherencia del ejemplo (que es bastante incoherente), podemos notar varias cosas interesantes:

1. Podemos usar a *self* en el bloque, y *self* sigue siendo pepita, no el bloque.
2. Podemos usar las variables locales del método, como energiaConDesgaste.
3. Podemos usar los atributos de pepita, como es la energia.

Además, si pepita tuviera un método como:

```
>come: unaCantidad
  [ ^energia := energia + unaCantidad ] value
   pepita vola: 5.
```

el circunflejo (AKA sombrerito, chirimbolo para arriba, retorno, ^) no hace que salga del bloque, sino del método que se está ejecutando, en este caso come:. De esta manera, la segunda línea del método nunca llegaría a ejecutarse :).

Para sumar a todo esto, podemos comentar que los bloques pueden llevar parámetros (hasta 4) e invocarlos con los mensajes value:, value:value:, etc... hasta 4. Por ejemplo:

```
bloqueQueHaceMagia := [ :unNumero :otro | (unNumero + 5 ) * 84  + otro].
bloqueQueHaceMagia value: 4 value: 0.  “retorna 756.
```

En el bloque, todo aquello que esta previo al pipe (|), es el listado de parámetros, separados por espacios y precedidos por dos puntos cada uno.

También pueden definirse variables locales al bloque, al igual que en los métodos, como:

```
[ | estoEsUnaVariableLocal | 1 + 1 ]
```

En este caso, se da cuenta que es una variable local y no son parámetros porque no hay nada con dos puntos (:) y porque hay 2 pipes. Igualmente, pueden combinarse, de manera que quede como:

```
[ :parametro | | variableLocal | 'bloqueQueNoHaceNada']
```

Fechas

Según la RAE (Real Academia Española), fecha se define como “tiempo en que ocurre o se hace algo”. Para nosotros, son objetos que tienen esa misma responsabilidad, indicarnos el tiempo. Las fechas no son tan fáciles de usar como todos los objetos mencionados anteriormente. Ya no podemos hacer como con los números: 1+2 y usar el 1 y el 2 así nomás. Las fechas debemos crearlas, y hay varias formas de hacerlo:

fechita := Date newDay: 26 monthNumber: 8 year: 2008. “Donde los parámetros son día, mes y año”
fechaDeHoy := Date today. “teniendo en cuenta que hoy es 26/8/08”
fechaDeAyer := Date yesterday.
fechaDeManiana := Date tomorrow.
fechaVieja := Date newDay: 31 monthIndex: 1 year: 1987

Y una vez que tenemos fechas, podemos preguntarles cosas como:

fechita day. “Retorna el dia del año, en este caso 239”
fechaDeHoy dayOfMonth “Retorna el dia dentro del mes, o sea, 26”
fechaDeManiana monthIndex “Retorna el numero de mes, en este caso 8”.
fechaVieja monthName “este retorna 'Enero'”

fechaDeHoy between: fechaDeAyer and: fechaDeManiana. “este se fija si la fecha de hoy esta entre la fecha de ayer y la fecha de mañana ;) “

fechaDeHoy yearsSince: fechaVieja. “este me retorna 21, porque a la fecha de hoy pasaron 21 años desde la fecha vieja”.

También podemos operar con fechas:

fechaDeHoy addDays: 5. “retorna una nueva fecha, 5 dias mas adelante que la fecha de hoy”.

fechaDeHoy substractDays: 365 “retorna una nueva fecha, 365 dias antes que hoy”.

fechaDeHoy substractDate: fechaDeAyer “una nueva fecha del resultado de restar a la fecha de hoy la fecha de ayer”

Como venimos haciendo con los otros tipos de objetos, los alentamos a buscar *Date* y ver qué otros mensajes pueden mandarle a las fechas.

Booleanos

Como todos sabemos (o deberíamos, a esta altura de la vida) los booleanos no son más que los valores true y false (verdadero y falso).

En Smalltalk, son objetos y podemos listar entre los mensajes que entienden estos:

- Operadores lógicos:

  true & false "Es el mensaje and binario"
  true and: [false] "Lo mismo que arriba pero en mensaje de palabra clave"
  (1>2) | (3=4) "Ese palito es un or"
  (1>2) or: [3=4] "El or palabra clave"
  (1>2) not. "La negación"

- Condicionales:

  ifTrue:
  (1>2) ifTrue: [pepita vola] "Y pepita no vuela porque 1 no es mayor a 2"

  ifFalse:
  (1>2) ifFalse: [pepita vola] "¡Ahora pepita si vuela porque es falso!"

  ifTrue:ifFalse:
  (1>2) ifTrue: [pepita vola] ifFalse: [pepita come:50] "Pero acá come 50 y el primer bloque ni se ejecuta."

  ifFalse:ifTrue:
  (1>2) ifFalse: [pepita vola] ifTrue: [pepita come:50] "Y aca vuelve a volar. El segundo bloque no se ejecuta."

  Tener en cuenta que tanto ifTrue:ifFalse como ifFalse:ifTrue: son un solo mensaje cada uno, que reciben dos parámetros respectivamente. No son dos mensajes "concatenados" ni cosas raras por el estilo.

Los booleanos también pueden ser investigados. Solo deben buscar *Boolean* en el *class browser*, y mirar los métodos en ese lugar.

nil[1]

Una variable, ¿puede no hacer referencia a nada? Y… no. Siempre hace referencia a un objeto. Entonces, ¿A qué objeto apuntan antes de inicializarse? Ese objeto es nil. Siempre que tengamos una variable sin inicializar, va a apuntar a nil, sea una variable de nuestro workspace, o una variable de un objeto nuestro.

Pero nil no es "nada", es un objeto que representa a la nada, a la nulidad. Es nil.
Y como nos preguntamos con cada uno de los anteriores. ¿Qué mensajes entiende nil?
Yyyyy... pocos, casi ninguno me atrevería a decir. Pero tiene otra particularidad. Responde de manera distinta a mensajes que entienden toooodos los objetos. Por ejemplo:

A cualquier objeto le podemos preguntar isNil o notNil:

2 isNil. "Este dice false"

[1] "Se acuerdan de nil? Volvió en forma de fichas!" – N. Scarcella

nil isNil. "Este me dice true"
2 notNil. "Sí, no es nil, da true, obvio"
nil notNil. "false, nil es nil :P"

También hay mensajes parecidos a los ifTrue: e ifFalse: como:

ifNil:, ifNotNil:, ifNil:ifNotNil:... Y reciben bloques, como los booleanos.

unaVariable ifNil:[pepita come:5000000] "Si la variable apunta a nil pepita va a comer mucho"

Conclusiones

Si bien los anteriores no son los únicos objetos básicos, consideramos que son los principales y más útiles. Si quieren, pueden investigar y profundizarlos buscando en el *class browser*, como esta dicho en todo este pequeño apunte, y hasta buscar otros objetos como *Time* y *TimeStamp*, entre otros.

La idea de aprender a manejar estos objetos es hacernos la vida más fácil a la hora de programar. ¡Ya no hace falta poner un '\0' para indicar el fin de una cadena!

Los invito a que se sienten a probar alguna que otra cosa y sean felices programando en este lenguaje tan bonito.