PROGRAMACIÓN ORIENTADA A OBJETOS

EL LENGUAJE SMALLTALK

GIPSI

TEMAS:

- ⇒ **Introducción a Smalltalk**
- ⇒ **Smalltalk: Conceptos básicos**
  - **Clase**
  - **Instancia**
  - **Mensaje**
  - **Herencia**
- ⇒ **Sintaxis del lenguaje**
  - **Expresiones**
  - **Literales**
    - ∗ **tipos**
  - **Variables**
    - ∗ **tipos**
    - ∗ **asignaciones**
    - ∗ **pseudovariables**
  - **Mensajes**
    - ∗ **tipos**
    - ∗ **evaluación de las expresiones de mensajes**
    - ∗ **mensajes en cascada**
    - ∗ **polimorfismo**
  - **Métodos y expresiones de retorno**
    - ∗ **tipos**
    - ∗ **estructura de cada tipo**
  - **Estructura de control: Selección o Alternativa**
    - ∗ **tipos**
    - ∗ **Mensajes de comparación a objetos**
  - **Bloques**
  - **Estructura de control: Repetición o Iteración**
    - ∗ **tipos**

Árbol genealógico de los lenguajes OO

Introducción a SMALLTALK

1. SMALLTALK es un lenguaje orientado a objetos puro, pues todas las entidades que maneja son objetos. El lenguaje se basa en conceptos tales como objetos y mensajes.

2. SMALLTALK es descendiente del lenguaje SIMULA y tiene sus orígenes en el Centro de Estudios de Palo Alto de Xerox, en los comienzos de 1970. Su desarrollo se basa en gran parte en las ideas de Alan Kay. Las tres versiones principales del lenguaje son SMALLTALK-72, SMALLTALK-76 y SMALLTALK-80.

3. SMALLTALK es mucho más que un lenguaje de programación, es un ambiente completo de desarrollo de programas. Éste integra de una manera consistente características tales como un editor, un compilador, un debugger, utilitarios de impresión, un sistema de ventanas y un manejador de código fuente.

4. SMALLTALK elimina la frontera entre aplicación y sistema operativo, modelando todos los elementos como objetos.

La programación en SMALLTALK requiere de al menos los siguientes conocimientos:

1. los conceptos fundamentales del lenguaje: manejo de clases y objetos, mensajes, clases y herencia.

2. la sintaxis y la semántica del lenguaje.

3. cómo interactuar con el ambiente de programación de SMALLTALK para construir nuevas aplicaciones SMALLTALK.

4. las clases fundamentales del sistema, tales como numéricas, colecciones, gráficas y las clases de interfase del usuario.

Diseñar nuevas aplicaciones SMALLTALK, requiere de conocimientos sobre las clases existentes en el sistema SMALLTALK. Frecuentemente la programación en SMALLTALK se denomina

"Programación por extensión"

Las nuevas aplicaciones son construidas por extensión de las librerías de clases de SMALLTALK.

SMALLTALK: CONCEPTOS BÁSICOS

Los conceptos básicos son:

- **Clase**
- **Instancia**
- **Mensaje**
- **Herencia**

La programación en SMALLTALK consiste en:

- **Crear clases.**
- **Crear instancias.**
- **Especificar la secuencia de mensajes entre objetos.**

CLASE

Es una colección de objetos que poseen características y operaciones comunes.

Una clase contiene toda la información necesaria para crear nuevos objetos y permite agrupar bajo un mismo nombre las variables y los métodos que manipulan esas variables.

Variables de la clase
Mét m1 Mét m2 Mét m3 ... Mét mn

Las variables de la clase son accedidas por los métodos m1, m2, m3, ... mn.

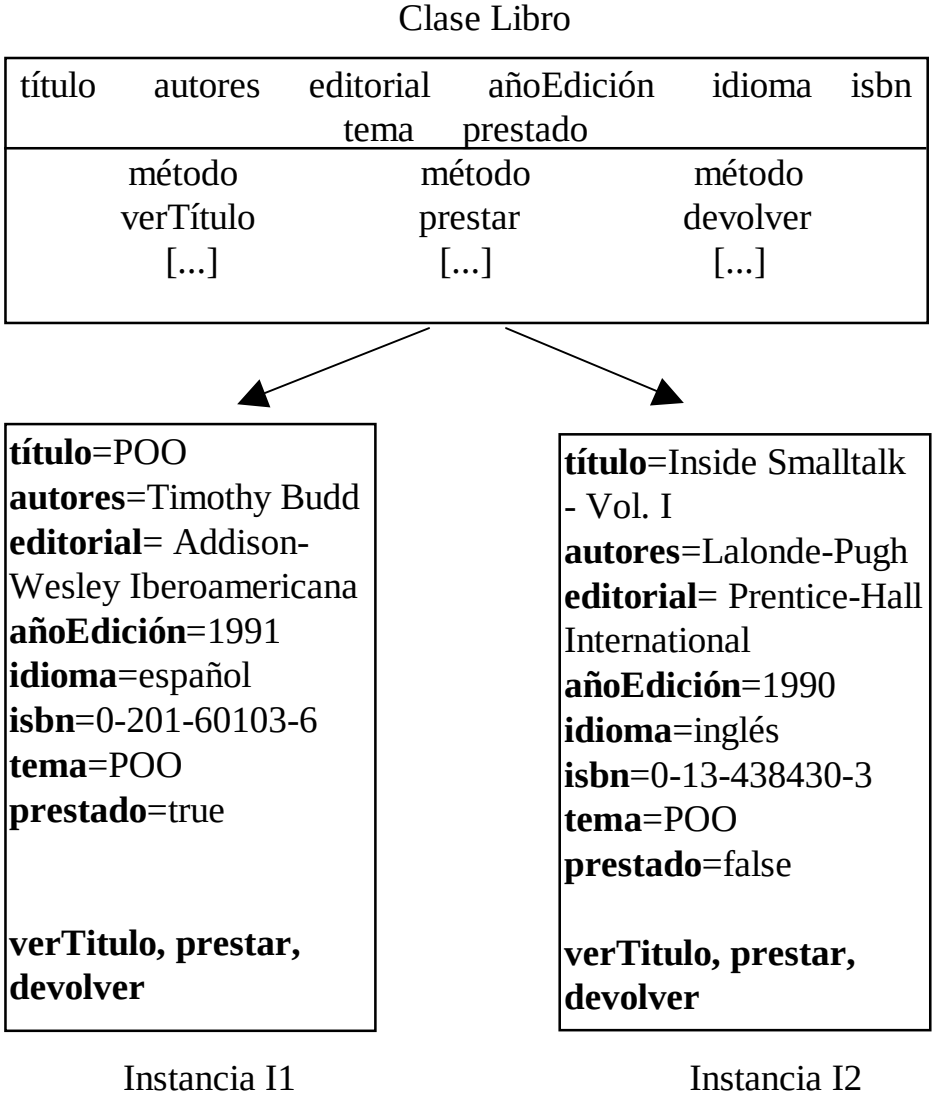
MÉTODO

Es el procedimiento local a una clase. Su activación permite el acceso a las variables de la clase.

INSTANCIA DE CLASES

A partir de una clase se pueden crear tantos elementos como se deseen. A estos elementos creados se los denomina Instancias u Objetos de la clase.

Por lo tanto dos instancias diferentes de una misma clase comparten los mismos métodos y la misma lista de variables con valores diferentes.



MENSAJES

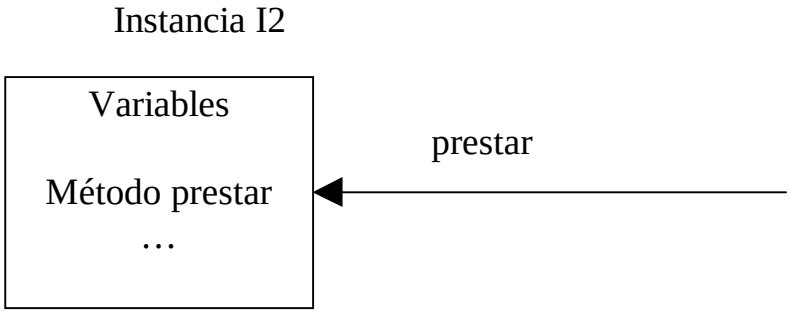
Es una petición a un objeto para que brinde algún servicio que el objeto puede realizar.

⇒ El mensaje especifica que operación se debe llevar a cabo, pero no cómo realizarla.

⇒ El objeto al que se envía el mensaje se denomina RECEPTOR del mensaje.

⇒ El texto del envío de un mensaje está compuesto de:

- el nombre del objeto destinatario denominado RECEPTOR DEL MENSAJE.
- un SELECTOR, que es el nombre del método (punto de entrada en el objeto receptor).
- y eventualmente, PARÁMETROS del método que se quiere activar, denominados ARGUMENTOS DEL MENSAJE.



Texto del envío del mensaje: **I2** **prestar**

selector

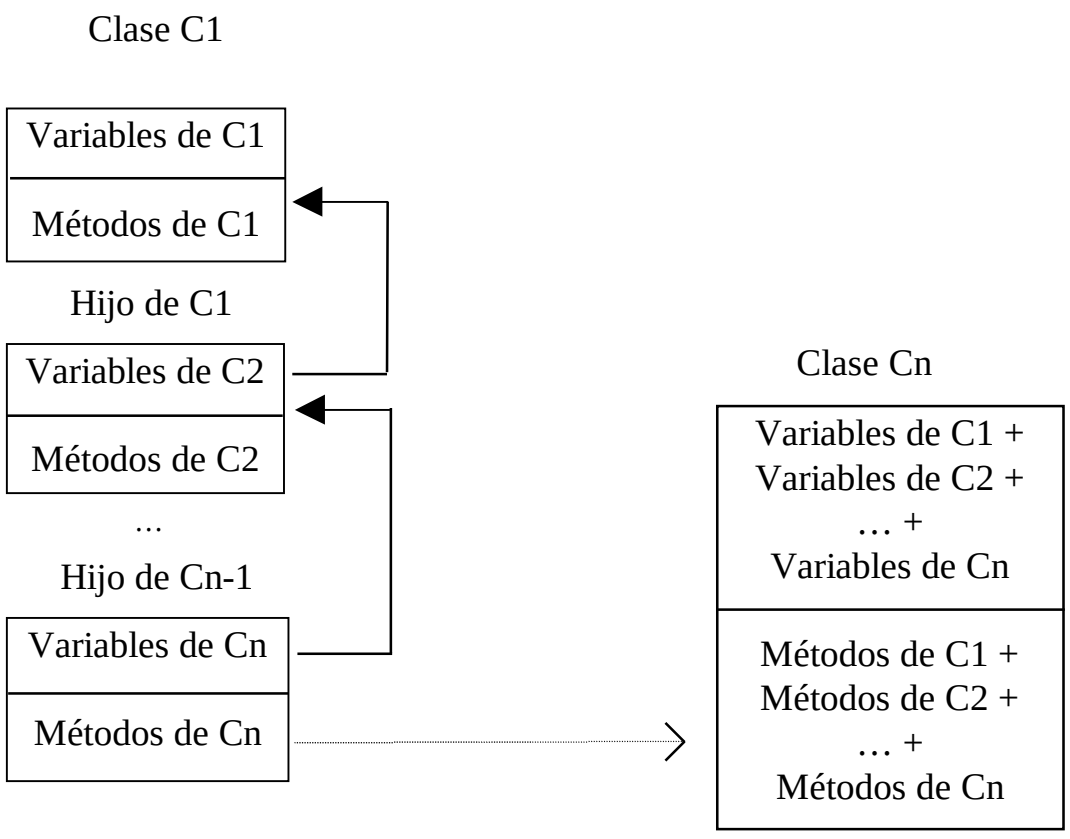
receptor

HERENCIA

⇒ Permite crear nuevas clases partiendo de otras previamente definidas con características semejantes a la que se quiere crear.

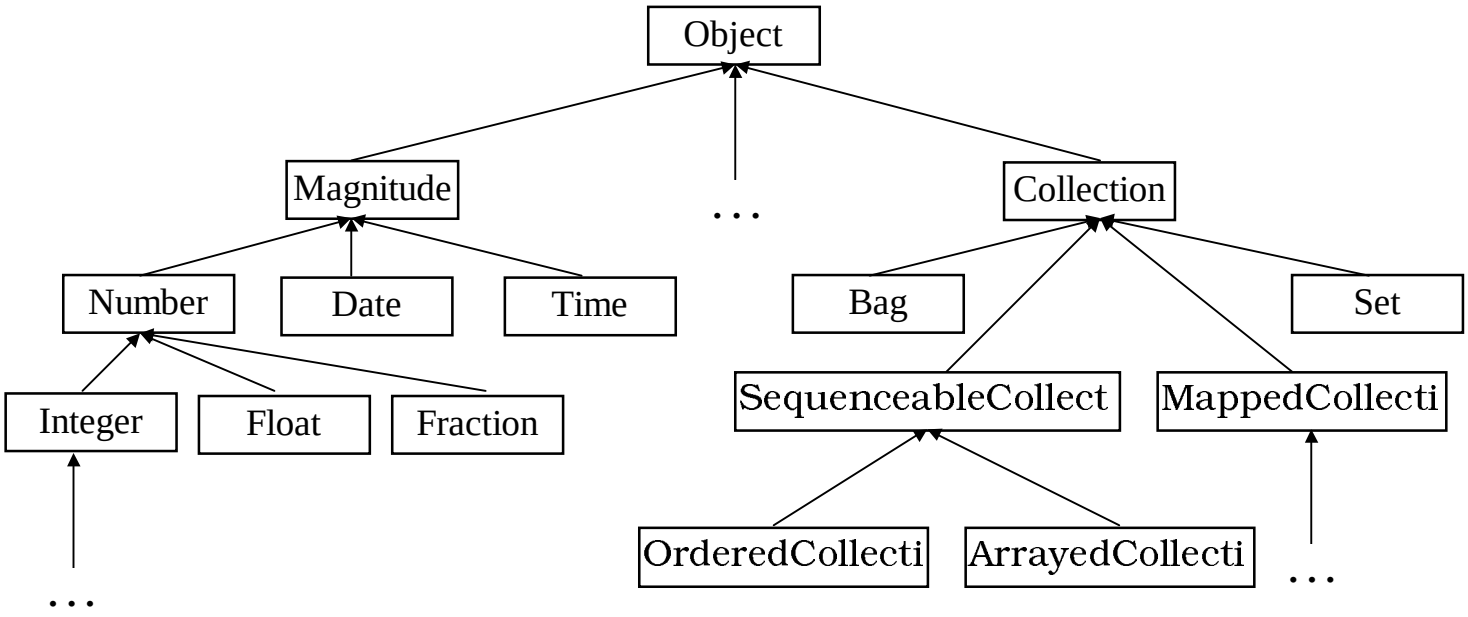
⇒ El mecanismo de herencia nos permite definir las propiedades particulares de un nuevo objeto y heredar las propiedades comunes ya existentes.

Herencia en la definición de la clase Cn



En SMALLTALK, todas las clases heredan de una única clase llamada Object.

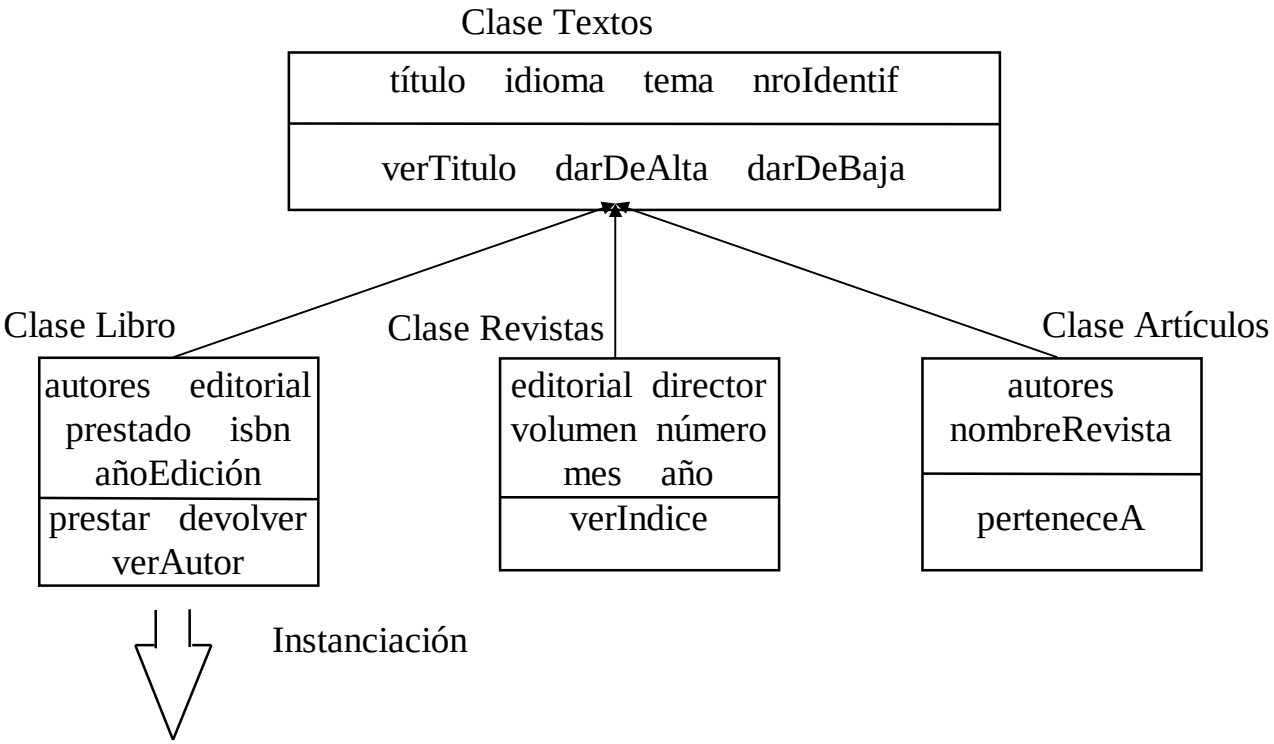
Representación ascendente (Bottom-up) parcial de las clases provistas por el Smalltalk



SMALLTALK maneja sólo HERENCIA SIMPLE.

La herencia se hace sobre las clases y NO sobre las instancias.

Una instancia de una clase maneja el conjunto de variables y métodos disponibles en dicha clase.



título=POO
autores=Timothy Budd
editorial= Addison-Wesley Iberoamericana
añoEdición=1991
idioma=español
isbn=0-201-60103-6
nroIdentif=194785
tema=POO
prestado=true

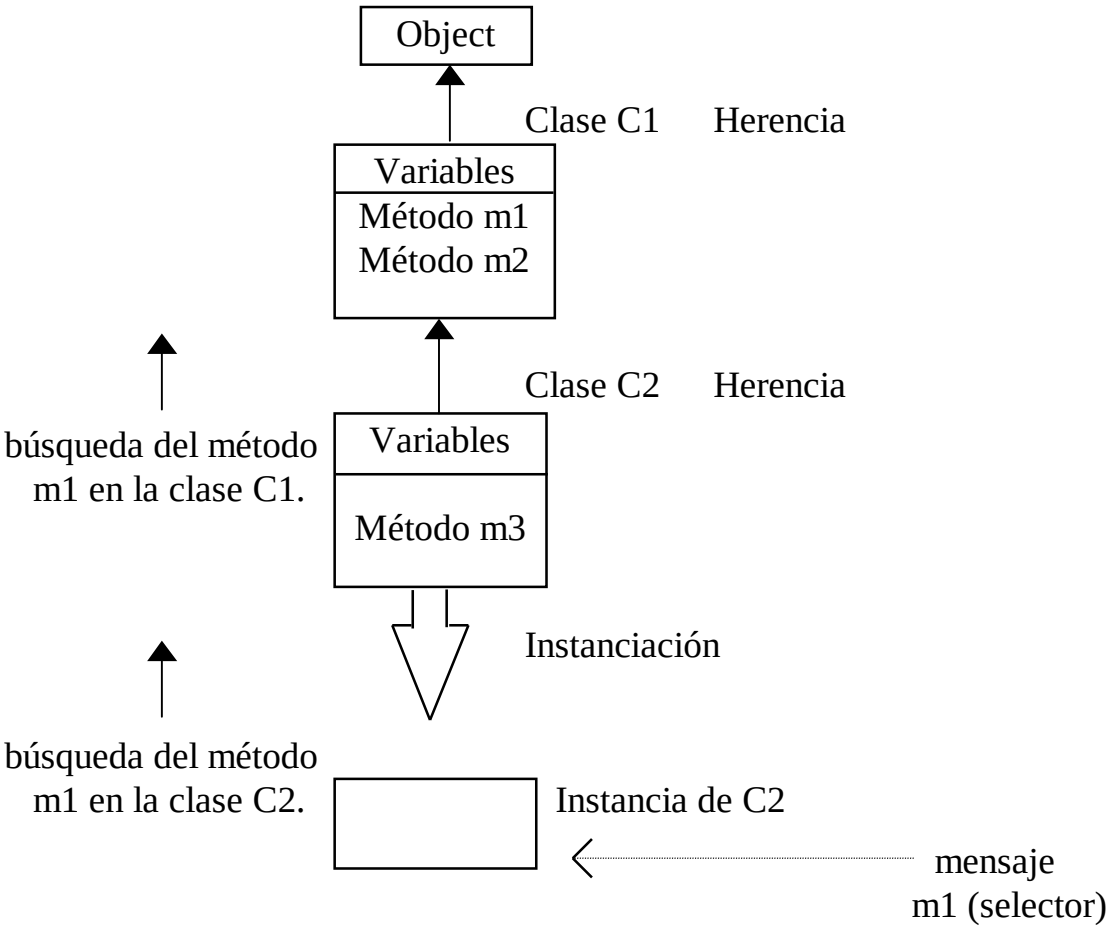
verTitulo, prestar, devolver, verAutor, darDeAlta, darDeBaja

ACTIVACIÓN DE MÉTODOS

Cuando una instancia recibe un mensaje, el selector del mensaje podrá o no corresponder con un método local a la clase de la instancia.

1. Si el selector corresponde a un método local de la clase del objeto, entonces hay activación de ese método.
2. Si el selector no corresponde a un método local a la clase del objeto, entonces se busca el método en la superclase de la clase del objeto y así recursivamente hasta encontrar la primera clase del árbol de herencia que posea un método idéntico al selector del mensaje.

Como vemos la investigación se hace de abajo a arriba.



SINTAXIS DE SMALLTALK

La sintaxis de SMALLTALK es:

objeto mensaje

Por ejemplo

- **3 factorial**

significa que el mensaje factorial es enviado al objeto 3 (instancia de la clase Integer) y por lo tanto se ejecutará el método de nombre factorial que se encuentra en la clase Integer o en sus superclases.

- **'HOLA' size**

- **#(4 8 6 3) at: 2**

- **$A asciiValue**

EXPRESIÓN

Smalltalk es un lenguaje basado en expresiones. Una expresión es una secuencia de caracteres que puede ser evaluada.

Hay cuatro tipos de expresiones:

1. **Literales**
2. **Nombres de Variables**
3. **Expresiones de mensajes**
4. **Expresiones de bloque**

La interfase de usuario permite la selección y evaluación de estas expresiones.

TIPOS DE EXPRESIONES

1- LITERALES (constantes literales u objetos constantes)

El valor de una expresión literal es siempre el mismo objeto.

Hay cinco tipos de constantes literales:

1. Números
2. Caracteres
3. Secuencia de caracteres
4. Símbolos
5. Arreglos

NÚMEROS

Descripción:	Los números son objetos que representan valores numéricos y responden a mensajes que calculan resultados matemáticos.
Representación:	Secuencia de dígitos precedidos o no de un signo '-' y/o con un punto decimal.
Ejemplos:	25 27.5 -35.7 -128 Notación científica: 25.53e2 -8.126e-3.

CARACTERES

Descripción:	Los caracteres son objetos que representan los símbolos que forman un alfabeto.
Representación:	Expresión precedida por el signo $ seguida por cualquier carácter
Ejemplos:	$a $A $3 $+ $$

SECUENCIA DE CARACTERES

Descripción:	Son objetos que representan una cadena de caracteres. Responden a mensajes para acceder a caracteres individuales, sustituir secuencias, compararlas con otras secuencias y concatenarlas.
Representación:	Secuencia de caracteres encerrados entre comillas (‘’).
Ejemplos:	‘Hola’ ‘secuencia de caracteres’ ‘Region 001’ Se pueden concatenar cadenas separándolas con coma: ‘Esto es’, ‘una tira’, ‘de caracteres’ equivale a: ‘Esto es una tira de caracteres’.

SÍMBOLOS

Descripción:	Son objetos que representan secuencias de caracteres utilizados como nombres en el sistema.
Representación:	Secuencia de caracteres alfanuméricos precedidos por el carácter '#'.
Ejemplos:	#simbolo #B450 #region Los nombres de clases en SMALLTALK son nombres simbólicos: #Figuras #Cuenta #Pila.

ARREGLOS

Descripción:	Son objetos estructurados cuyos elementos son accesibles mediante un índice entero. Cada elemento del array es un literal. Estos objetos responden a mensajes que piden acceso a su contenido.
Representación:	Secuencia de literales separados por blancos y encerrados entre paréntesis y precedidas por el carácter '#'. Los símbolos y arreglos contenidos como literales en el arreglo, no se preceden con el carácter '#'
Ejemplos:	#(1 2 3) arreglo de tres elementos enteros. #('producto' '120' 'factura' 'cantidad') arreglo de cuatro elementos que son secuencias de caracteres. #('elemento' (1 $4 'uno') 4 $A) arreglo de cuatro elementos: secuencia, array, entero y carácter.

2- VARIABLES Y ASIGNACIÓN

TODA VARIABLE EN SMALLTALK ES UN OBJETO PUNTERO QUE PERMITE REFERENCIAR OTRO OBJETO

Los nombres de variables en SMALLTALK son identificadores que consisten en una secuencia de letras y dígitos que comienza con una letra.

Una expresión de asignación consta de la variable cuyo valor va a cambiarse, seguido de un prefijo de asignación: una flecha apuntando hacia la izquierda (‘←’) en SMALLTALK-80 o un ‘:=‘ en SMALLTALK V.

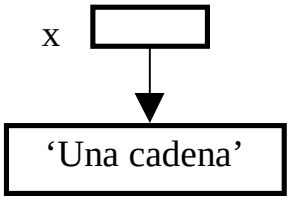
variable ← expresión	**(SMALLTALK-80)**
variable := expresión	**(SMALLTALK V)**

VARIABLES Y ASIGNACIÓN

EJEMPLOS

x ← ‘Una cadena’

la variable x apunta al objeto ‘Una cadena’



Una variable puede contener diferentes punteros a objetos a lo largo de la ejecución

x ← **‘Una cadena’**

...

x ← **23 factorial**

...

x ← **Array new: 5**

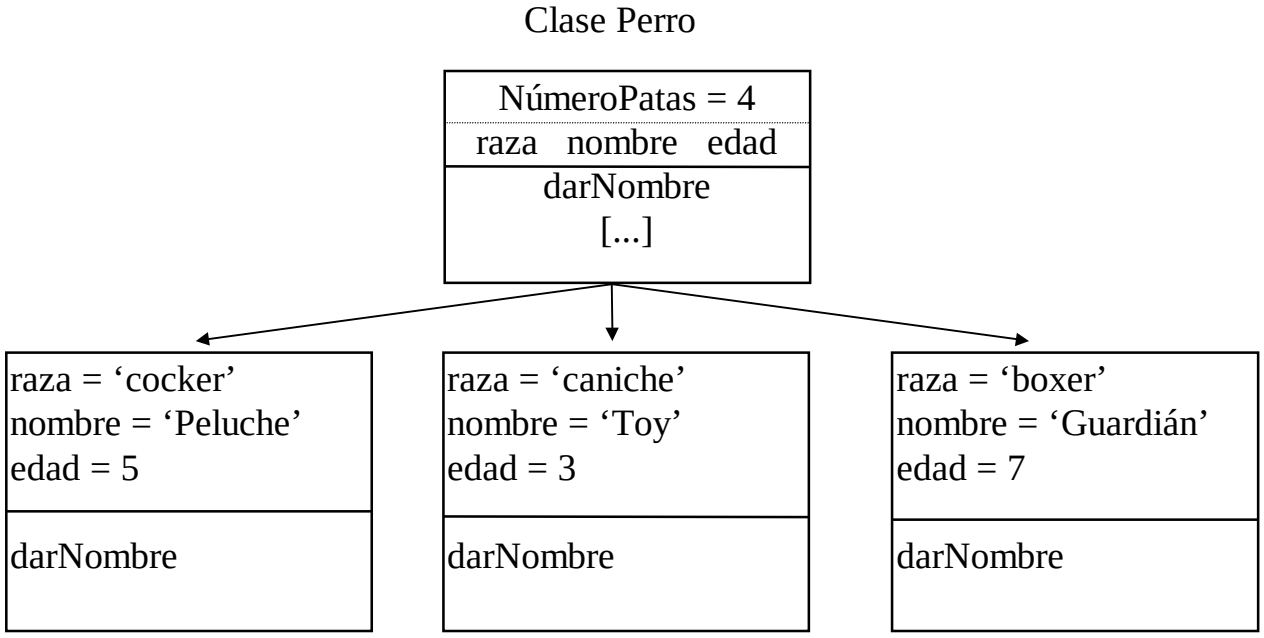
TIPOS DE VARIABLES

Hay dos tipos de variables:

1. **variables privadas.**
2. **variables compartidas**

Las **VARIABLES PRIVADAS** sólo son accesibles por un objeto.

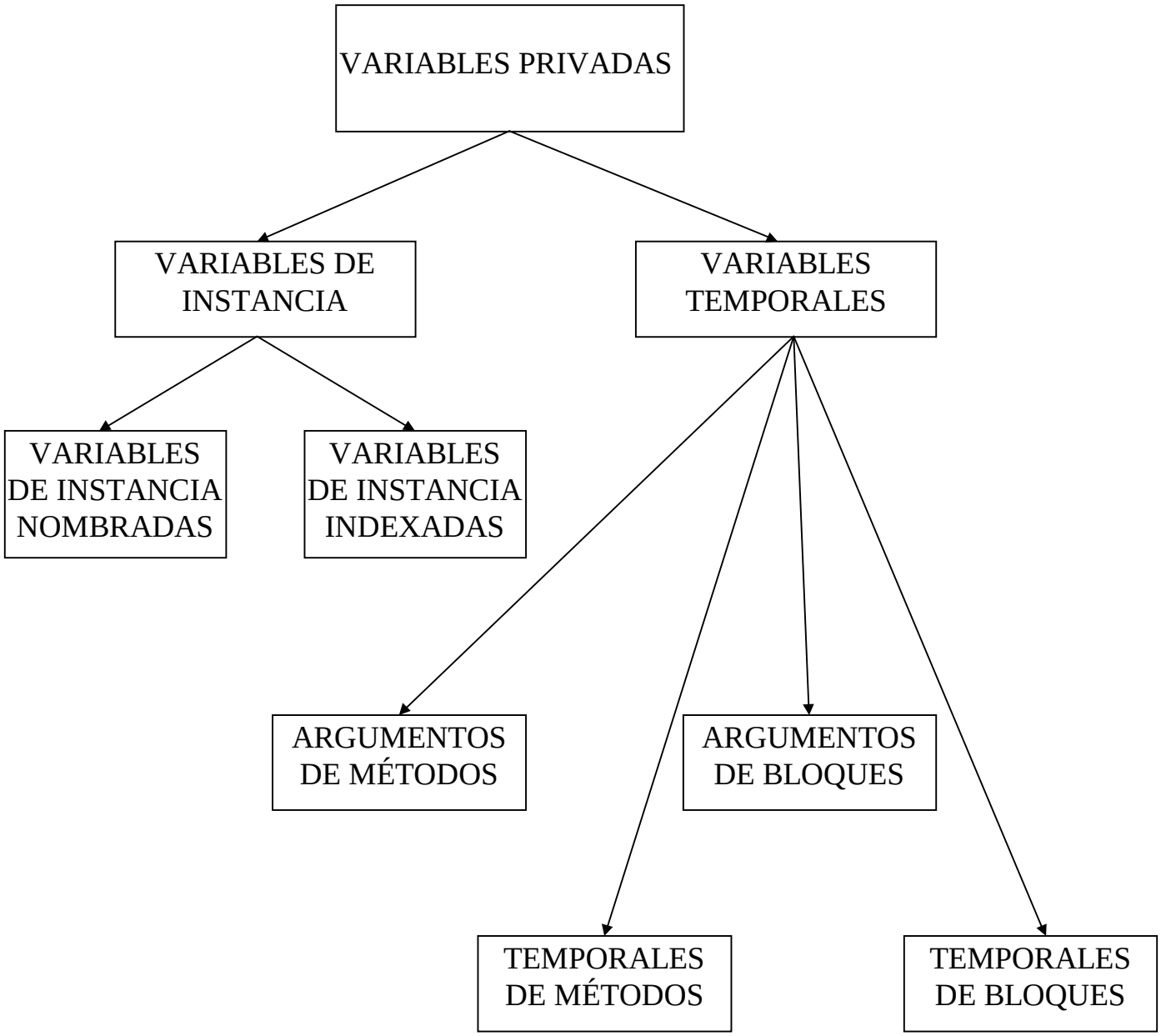
Las **VARIABLES COMPARTIDAS** pueden ser accedidas por más de un objeto.



Los nombres de las variables privadas deben comenzar con una letra minúscula y los nombres de las variables compartidas deben comenzar con letra mayúscula.

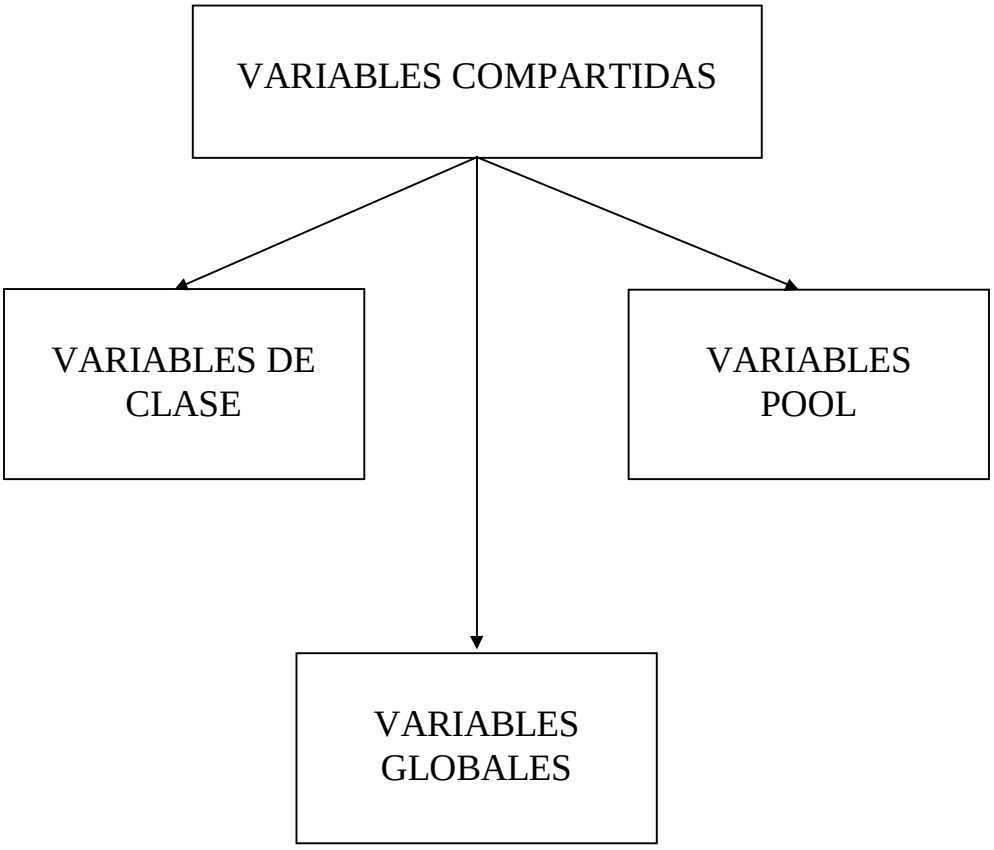
VARIABLES PRIVADAS

CLASIFICACIÓN



VARIABLES COMPARTIDAS

CLASIFICACIÓN



VARIABLES PRIVADAS

VARIABLES DE INSTANCIA

⇒ Aunque todas las instancias de una clase tienen el mismo conjunto de variables de instancia, sus valores son únicos a cada una de las mismas.

⇒ Sólo pueden ser accedidas en métodos de instancia disponibles en la clase.

Las variables de instancia existen durante todo el tiempo de vida de un objeto y representan el estado del objeto.

1. Variables de Instancia Nombradas

Son variables que tienen asociado un nombre y son identificadas por él.

2. Variables de Instancia Indexadas

Son variables que no tienen nombre y sólo pueden accederse enviando un mensaje a la instancia con un índice que especifique a cual de las variables indexadas se desea acceder.

Los dos mensajes principales de indexación son: **at:**, **at:put:**.

EJEMPLO:

arreglo ← Array **new:** 5	Retorna una instancia de la clase Array.
arreglo **at:** 1 **put:** 'abc'	La primera variable de instancia de arreglo es inicializada con la cadena 'abc'.
arreglo **at:** 1	Retorna el valor de la primera variable de instancia: 'abc'.

VARIABLES PRIVADAS (continuación)

VARIABLES TEMPORALES

⇒ Son las variables que están definidas en los métodos, bloques o programas Smalltalk. Se utilizan en una actividad dada y después se destruyen.

⇒ Representan un estado transitorio de un objeto, existen mientras dura la activación del método, bloque o programa.

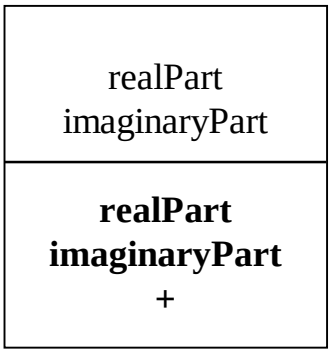
EJEMPLOS

1. Uso de variables temporales en un programa:

```
| complejo1 complejo2 suma |

complejo1 ← Complex newWithReal: -5.9 andImaginary: 4.3.

complejo2 ← Complex newWithReal: 8.5 andImaginary: 1.0.

suma ← complejo1 + complejo2.

...
```

EJEMPLOS (continuación)

Clase Complex



2. Uso de variables temporales de método y parámetros de método en el método de instancia + de la clase Complex:

+ unComplejo

"Retorna una instancia de la clase Complex igual a la suma del receptor y el argumento unComplejo"

|sumaParteReal sumaParteImaginaria|

sumaParteReal ← realPart + unComplejo **realPart**.

sumaParteImaginaria ← imaginaryPart + unComplejo **imaginaryPart**.

...

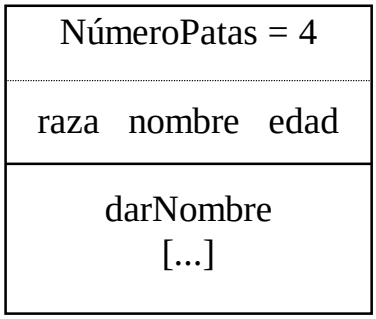
VARIABLES COMPARTIDAS

VARIABLES DE CLASE

⇒ Son variables compartidas por todas las instancias de una clase y sus subclases. Tienen el mismo valor para todas las instancias. Sólo pueden ser accedidas por los métodos de clase y los métodos de instancia de la clase y sus subclases.

EJEMPLO

Clase Perro



Object **subclass:** #Perro
instanceVariableNames:
'raza nombre edad'
classVariableNames:
'NumeroPatas'
poolDictionaries:
''

PSEUDOVARIABLES

Existen cinco pseudovariables:
nil, true, false, self y super.

⇒ El valor de una pseudovariable no puede cambiarse con una expresión de asignación.

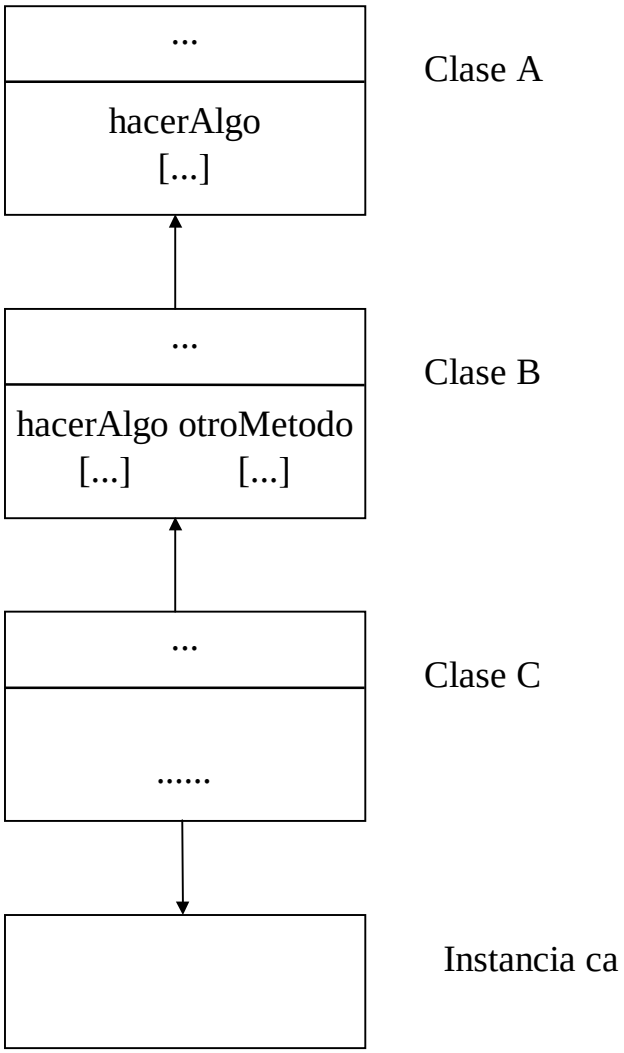
⇒ **nil, true y false** son instancias de clases.

⇒ **self** y **super** son objetos cuyo significado depende del contexto donde se usen.

⇒ Las cinco son palabras reservadas y son globales.

nil	es el valor por defecto de una variable. Es usado para representar el valor de objetos no inicializados.
true	es la única instancia de la clase True y representa el valor booleano VERDADERO.
false	es la única instancia de la clase False y representa el valor booleano FALSO.
self	es usada en métodos; su valor es siempre el objeto que recibe el mensaje que causa que el método que contiene **self** sea ejecutado.
super	es usada en métodos. Su valor es equivalente a **self,** el objeto que recibe el mensaje que causa que el método que contiene **super** sea ejecutado, pero la estrategia de búsqueda del método es diferente. La búsqueda comienza en la **superclase** inmediata de la **clase** que contiene el método en el cual **super** aparece**.** Los mensajes a **super** son utilizados cuando se quiere utilizar un método de una **superclase** que es redefinido en una **subclase**.

USO DE SELF Y SUPER



Texto del envío del mensaje: **ca *otroMetodo***.

Posibles implementaciones del método de instancia *otroMetodo*:

OtroMetodo ... self **hacerAlgo**. (ejecuta definición realizada en la clase B)	***OtroMetodo*** ... super **hacerAlgo**. (ejecuta definición realizada en la clase A)

3- EXPRESIONES DE MENSAJES

Las expresiones de mensajes en SMALLTALK describen quién es el receptor del mensaje, el nombre del mensaje y los argumentos.

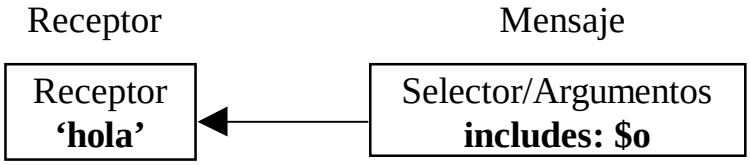
Por ejemplo:

16.79 **rounded**

'hola que tal' **size**

'hola' **includes:** $o

numerador **+** (unaFracción **verNumerador**).



Por ejemplo en la expresión SMALLTALK 'hola' **includes:** $o, la cadena **'hola'** es el receptor del mensaje; **includes:** es el selector (nombre del mensaje) que identifica unívocamente la operación que es seleccionada y **$o** es el argumento necesario para realizar la operación.

La expresión es evaluada de la siguiente manera:

1. el mensaje **includes: $o** es enviado al objeto cadena **'hola'**.
2. el objeto cadena sabe como responder a este mensaje y el objeto booleano true es retornado como un resultado.

TIPOS DE MENSAJES

SMALLTALK soporta tres tipos primitivos de mensajes:

- UNARIOS (unary)

- BINARIOS (binary)

- de PALABRA CLAVE (keyword)

MENSAJES UNARIOS

Los mensajes unarios no tienen argumentos, su sintaxis comprende solamente un receptor y un selector.

EJEMPLOS

7 isPrime:	el mensaje consistente del selector **isPrime** es enviado al objeto entero **7**. El objeto **true** es retornado como el resultado.
$a asInteger	el mensaje consistente del selector **asInteger** es enviado al objeto carácter **a**. El objeto entero representando el valor ordinal del carácter es retornado como el resultado.
car isVowel	el mensaje consistente del selector **isVowel** es enviado al objeto **variable car**. El objeto **true** o **false** es retornado como resultado.
Carta new	el mensaje consistente del selector **new** es enviado a la **clase Carta**. Crea una nueva instancia de la clase Carta.

MENSAJES BINARIOS

⇒ Además del receptor y selector, la sintaxis de los mensajes binarios tiene un único argumento. Los selectores para los mensajes binarios son caracteres especiales simples o dobles.

⇒ Los selectores de carácter simple incluyen operadores de comparación y aritméticos tales como:

+ - * / < > =

⇒ Los selectores de carácter doble incluyen operadores tales como

~= (not =) <= // (división entera)

EJEMPLOS

1.5 + 6.3e2 — el mensaje **+ 6.3e2** es enviado al objeto float **1.5**. El selector es + y el argumento es el objeto 6.3e2. El resultado retornado es el objeto float **631.5**.

'abc' ~ = 'def' — el mensaje **~ = 'def'** es enviado al objeto cadena **'abc'**. El selector es **~ =** y el argumento es el objeto cadena 'def'. El receptor y el argumento cadena son comparados por desigualdad. El objeto booleano **true** es retornado.

'hola ','que tal' — el mensaje **, 'que tal'** es enviado al objeto cadena **'hola'**. El selector es **,** y el argumento es el objeto 'que tal'. El objeto cadena 'hola que tal' es retornado.

MENSAJES DE PALABRA CLAVE

⇒ Estos mensajes contienen una o más palabras claves, donde cada palabra clave tiene un argumento simple asociado con ella.

⇒ El nombre de la palabra clave siempre termina en dos puntos (:). Los dos puntos son parte del nombre (éste no es un terminador especial).

⇒ El selector en un mensaje de más de una palabra clave se forma concatenando todas las palabras claves, por ejemplo **between:and:**, **at:put:**, etc.

MENSAJES DE PALABRA CLAVE

EJEMPLOS

28 gcd: 12 — el mensaje **gcd: 12** es enviado al objeto entero **28**, el selector es gcd: y el argumento es el objeto entero 12. El resultado que se retorna es el máximo común divisor entre el objeto receptor 28 y el objeto argumento 12, es decir, el objeto **4**.

#(4 3 2 1) at: 4 — el mensaje **at: 4** es enviado al objeto arreglo **(4 3 2 1)**. El selector es at: y el argumento es el objeto entero 4. El resultado retornado es el objeto entero **1**, el objeto asociado con el índice 4 en el arreglo.

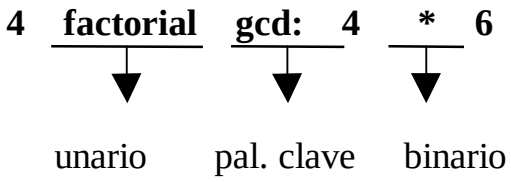
vector at: 2 put: ‘st’ — El mensaje **at: 2 put: ’st‘** es enviado a la **variable vector**, el selector es at:put: y los argumentos son los objetos 2 y ‘st’. El resultado es colocar el objeto cadena ‘st’ como segundo elemento en el objeto arreglo apuntado por la variable vector.

Array new: 3 — El mensaje **new: 3** es enviado a la **clase Array**. El selector es new: y el argumento es el objeto entero 3. El resultado retornado es una instancia de la clase **Array** de 3 elementos.

EVALUACIÓN DE LAS EXPRESIONES DE MENSAJES

El receptor o argumento de una expresión de mensajes puede ser en si mismo una expresión de mensajes. Esto da origen a expresiones complejas de mensajes y a la necesidad de un orden de evaluación.

Por ejemplo, el siguiente mensaje contiene mensajes de los tres tipos:



En SMALLTALK la relación de precedencia en la evaluación de expresiones es la siguiente:

1. **expresiones entre paréntesis.**
2. **expresiones unarias (evaluadas de izquierda a derecha).**
3. **expresiones binarias (evaluadas de izquierda a derecha).**
4. **expresiones de palabra clave.**
5. **expresiones de asignación.**

EVALUACIÓN DE LAS EXPRESIONES DE MENSAJES

EJEMPLOS

EXPRESIONES DE MENSAJES		RESULTADOS DE LA EVALUACIÓN
2 factorial negated	⇨	
3 + 4 * 6 + 3	⇨	
2 / 3 * 5 / 7	⇨	
2 / (3 * 5) / 7	⇨	
15 gcd: 32 // 2	⇨	
2 factorial + 4	⇨	
5 between: 1 and: 16 squared + 4	⇨	
Resul ← 4 factorial gcd: 4 * 6	⇨	
#(1 2 $a), #($b ‘cd’)	⇨	
‘maría’ at: 1 put: $M	⇨	

MENSAJES EN CASCADA

⇒ Un mensaje en cascada es una forma práctica de especificar que varios mensajes se envían al mismo receptor.

⇒ En el mensaje en cascada se escribe el receptor seguido por la lista de expresiones de mensajes separados por punto y coma (;).

Por ejemplo, si se desea modificar los tres primeros elementos del objeto arreglo unArreglo, debe usarse el mensaje **at:** indice **put:** unValor, que modifica el elemento indice de unArreglo.

La expresión de la izquierda envía el mismo mensaje a unArreglo tres veces. Alternativamente se puede emplear la expresión de la derecha que tiene los mensajes en cascada.

UnArreglo at: 1 put: 3.
UnArreglo at: 2 put: 8.
UnArreglo at: 3 put: 5.

unArreglo at: 1 put: 3; at: 2 put: 8; at: 3 put: 5.

Conjunto ← Set new.
Conjunto add:1.
Conjunto add:2.
Conjunto add:3.

Set new add: 1; add: 2; add: 3.

El empleo de cascadas elimina, a menudo, la necesidad de las variables temporales.

POLIMORFISMO

(VINCULACIÓN DINÁMICA Y SOBRECARGA DE MENSAJES)

El mismo mensaje puede ser interpretado de diferentes formas por diferentes objetos. Por ejemplo:

5 + 100

(200 @ 200) + 100

Ambos usan el mensaje + 100, pero los objetos receptores reaccionan al mensaje de maneras muy diferentes.

En el primer ejemplo el receptor es el objeto **entero 5** y el selector **+** es interpretado como la **suma de enteros**. En el segundo ejemplo el receptor es el objeto **punto (x, y)** con coordenadas (200, 200). En esta expresión el selector **+** es interpretado como la **suma definida sobre puntos**. El punto con coordenadas (x, y) igual a 300 es retornado.

Es el receptor del mensaje el que determina como es interpretado el mensaje

El método suma realmente invocado por una expresión tal como unObjeto + 100 es determinado por el tipo de objeto que recibe el mensaje en el **tiempo de ejecución**. Esto se denomina **vinculación dinámica**.

Cuando el mismo selector es aceptado por clases diferentes de objetos, se dice que el selector está **sobrecargado**.

POLIMORFISMO

EJEMPLO

El método between:and: trabaja con varios tipos de argumentos.

El método between:and: está definido en la clase abstracta Magnitude y prueba si el receptor está entre dos extremos. Efectúa la comprobación por medio del envío del mensaje <=.

x **between:** 2 **and:** 4.	mensaje **between:and:** enviado a un objeto entero.
x **between:** $a **and:** $z.	mensaje **between:and:** enviado a un objeto carácter.
x **between:** 2@4 **and:** 12@14.	mensaje **between:and:** enviado a un objeto punto.
'carbon' **between:** 'carbohidrato' **and:** 'carbonato'.	mensaje **between:and:** enviado a un objeto cadena.

```
y ← 2.
z ← $d.
(y between: 1 and: 3) and:
    (z between: $b and: $t).
```

5 - EXPRESIONES DE BLOQUES

⇒ Son instancias de la clase BlockContext y representan una secuencia de expresiones separadas por puntos y encerradas entre corchetes '[]' que serán evaluadas sólo cuando el mensaje value adecuado sea enviado al bloque.

⇒ La evaluación de un bloque retorna el resultado de la evaluación de la última expresión del mismo.

Sintaxis:

```
[objeto1 mensaje1.
objeto2 mensaje2.
...]
```

⇒ Los bloques, como cualquier otro objeto, pueden ser asignados a variables y definidos en métodos o programas Smalltalk.

EXPRESIONES DE BLOQUES (continuación)

⇒ Los bloques pueden tener parámetros (variables temporales argumentos de bloque) y variables locales (variables temporales de bloque).

Sintaxis:

```
[:parámetro1 :parámetro2 :parámetro3 ... |
|local1 local2 ... localn|
objeto1 mensaje1.
objeto2 mensaje2.
...]
```

Según si los bloques tengan o no parámetros, se utilizan distintos mensajes value:

⇒ unBloque **value**

⇒ unBloque **value**: parámetro

⇒ unBloque **value**: parámetro1 **value**: parámetro2

⇒ unBloque **value**: parámetro1 **value**: parámetro2 **value**: parámetro3

⇒ unBloque **valueWithArguments**: unArregloDeParámetros

EXPRESIONES DE BLOQUES

EJEMPLOS

[‘evaluación retardada’ **size**] **value**.	**Bloque con una expresión de mensaje**. Se evalúa la expresión y es retornado el objeto entero 20 que representa en tamaño en caracteres del objeto cadena ‘evaluación retardada’.
[Date **today dayName**. ‘cadena a convertir’ **asUpperCase**.] **value**.	**Bloque con más de una expresión de mensaje**. Se evalúan las expresiones una a una y es retornado el objeto cadena ‘CADENA A CONVERTIR’ que representa el resultado de la evaluación de la última expresión del bloque.
| unBloque| unBloque ← [$o **isVowel**. ‘9 Sep 1995’ **asDate dayName**.] . ... unBloque **value asUpperCase**.	**Asignación de un bloque a una variable temporal**. En este caso el bloque es evaluado cuando se envía el mensaje value al objeto “apuntado” por unBloque, ya que el contenido de esta variable es el código a evaluar. El resultado es el objeto cadena que corresponde al día de la semana de la fecha utilizada, al que luego se le envía el mensaje para que sea convertida en mayúsculas.

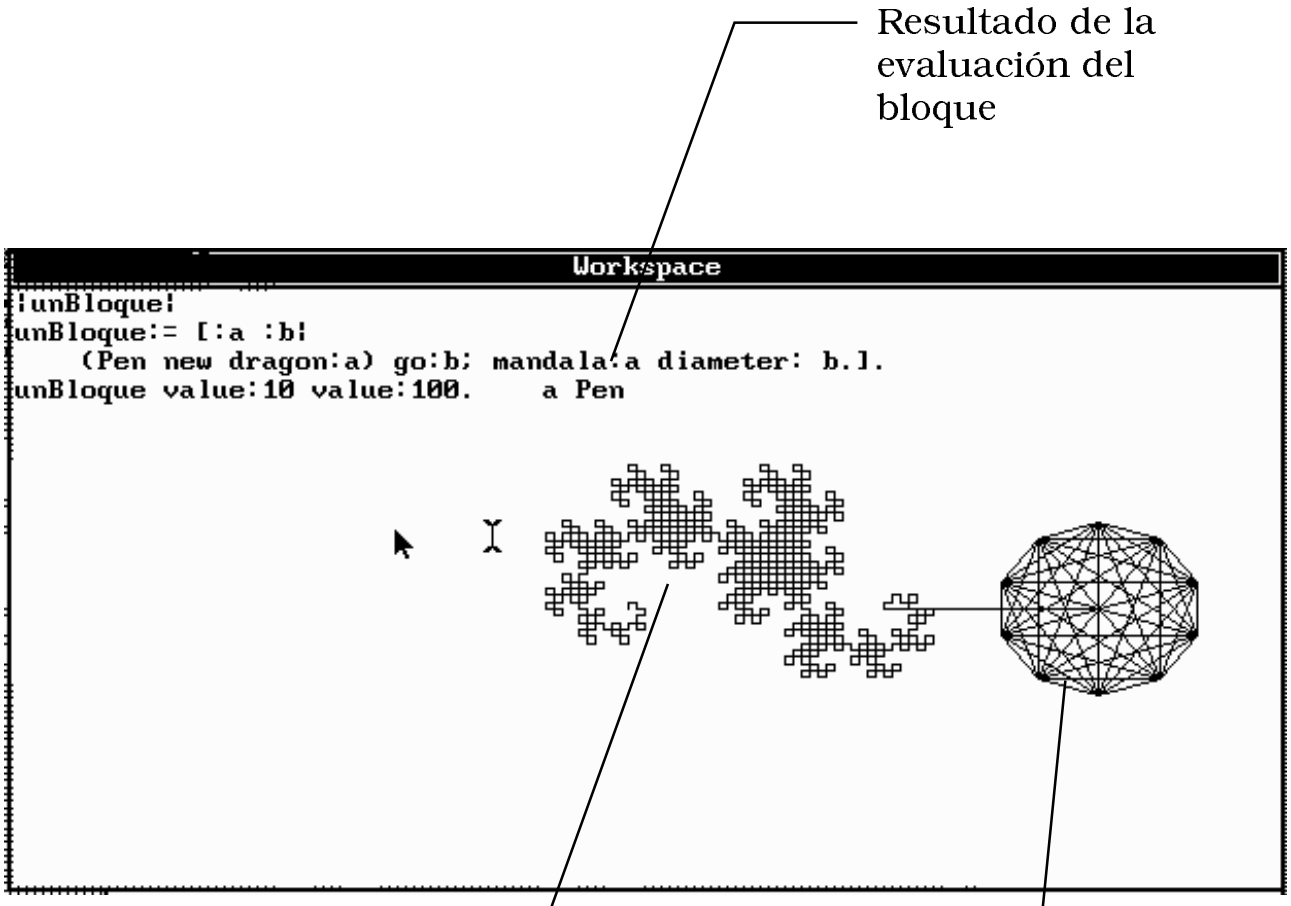
EXPRESIONES DE BLOQUES

EJEMPLOS (continuación)

```
| unBloque|
unBloque ← [ :a :b |
    (Pen new dragon: a)
        go: b;
        mandala: a diameter: b.
] .
...
unBloque value: 10 value: 100.
```

Asignación de un bloque con argumentos a una variable temporal. Como en el caso anterior el bloque es evaluado en el momento en el que se le envía el mensaje **value:** 10 **value:** 100. Dentro del bloque se crea una instancia de la clase Pen y se le envía el mensaje **dragon**: a. Luego a la instancia de la clase Pen se le envían en cascada los mensajes **go**: b y **mandala**: a **diameter**: b. (Ver figura 1)

FIGURA 1: VENTANA DE TRABAJO DE SMALLTALK V



MÉTODOS Y EXPRESIONES DE RETORNO

⇒ Los métodos de una clase son los procedimientos que se activan ante la recepción de un mensaje.

⇒ La forma en que un objeto responde a un mensaje está descripta en un Método. Cada clase contiene la lista de métodos que le permiten a sí misma y a sus instancias responder a los mensajes que le son enviados.

MÉTODOS: CLASIFICACIÓN

Los mensajes pueden ser enviados a las clases o las instancias. Por lo tanto los métodos que se activan ante dichos mensajes también deberán estar asociados a las clases o a las instancias:

⇒ **MÉTODOS DE INSTANCIA**: contienen los detalles de implementación para mensajes a los cuales sólo las instancias de una clase pueden responder.

⇒ **MÉTODOS DE CLASE**: contienen los detalles de implementación para mensajes a los cuales sólo las clases pueden responder. Generalmente los métodos de clase permiten: inicializar variables de clase, crear instancias de la clase, etc.

SINTAXIS DE UN MÉTODO SIN ARGUMENTOS

(MENSAJE UNARIO)

Nombre "Comentario" |Variables Temporales| Cuerpo del Método

⇒ El nombre de un método es el mismo que el nombre del selector del mensaje, al cual el método debe responder.

⇒ Es usual incluir unas líneas de comentarios entre comillas dobles ("") describiendo el método.

⇒ Seguidamente hay que declarar las variables temporales que se van a usar.

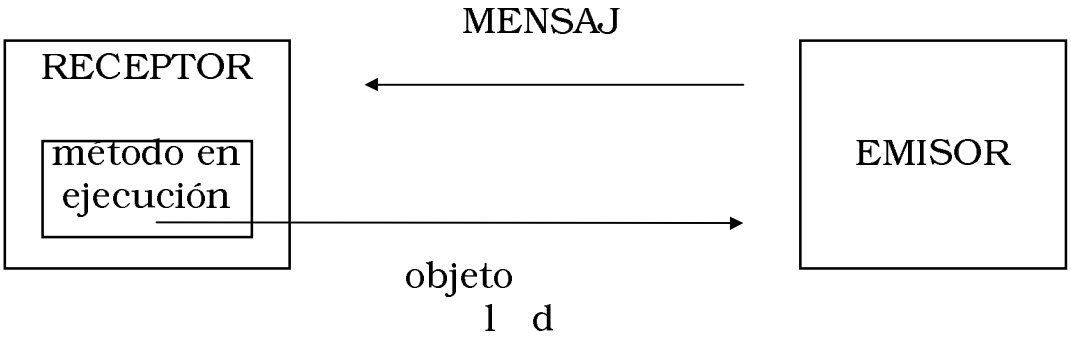
⇒ Estas variables son accesibles únicamente dentro del método y se destruyen tras la ejecución de él. Finalmente, el cuerpo del método está constituido por una secuencia de expresiones de mensajes separadas por puntos (.).

SINTAXIS DE UN MÉTODO SIN ARGUMENTOS

(continuación)

⇒ Cuando un método termina el receptor devuelve un objeto (resultado) al emisor del mensaje.

⇒ El resultado por defecto es el receptor mismo del mensaje.



⇒ Opcionalmente, se pueden utilizar Expresiones de Retorno para devolver otro objeto.

Una **expresión de retorno** es una expresión precedida por:

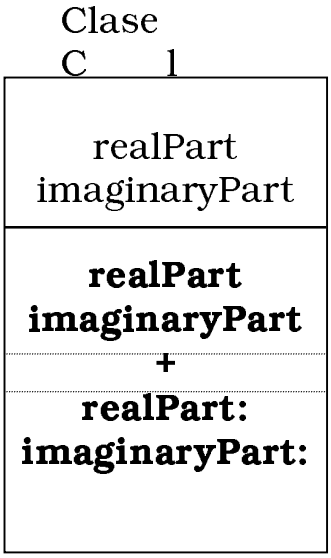
↑	**(SMALLTALK-80)**
^	**(SMALLTALK V)**

La evaluación de cualquier expresión de retorno termina automáticamente con la ejecución del método y devuelve el

resultado de la evaluación de la expresión precedida por el operador de retorno.

SINTAXIS DE UN MÉTODO SIN ARGUMENTOS

EJEMPLOS



Implementación de los métodos realPart e imaginaryPart de la clase complex

realPart

"Retorna la componente real del complejo receptor"

↑realPart

imaginaryPart

"Retorna la componente imaginaria del complejo receptor"

↑imaginaryPart

SINTAXIS DE MÉTODOS CON ARGUMENTOS

SINTAXIS DE UN MÉTODO CON UN ARGUMENTO (MENSAJE BINARIO)

Selector y Argumento "Comentario" |Variables Temporales| Cuerpo del Método

Se colocará el selector del mensaje seguido de un argumento (parámetro formal).

EJEMPLO

Implementación del método + de la clase complex

+ unComplejo

"Retorna una instancia de la clase Complex igual a la suma del receptor y el argumento unComplejo"
|sumaParteReal sumaParteImaginaria|
sumaParteReal ← realPart + unComplejo **realPart**.
sumaParteImaginaria ← imaginaryPart + unComplejo **imaginaryPart**.
...

SINTAXIS DE MÉTODOS CON ARGUMENTOS

(continuación)

SINTAXIS DE UN MÉTODO CON UNO O MÁS ARGUMENTOS (MENSAJE DE PALABRA CLAVE)

```
palabraClave1: argumento1  palabraClave2: argumento2 …
"Comentario"
|Variables Temporales|
Cuerpo del Método
```

Se colocarán cada una de las palabras claves que forman el selector y siguiendo a cada palabra clave el carácter ':' y un argumento formal.

SINTAXIS DE UN MÉTODO CON UNO O MÁS ARGUMENTOS (MENSAJE DE PALABRA CLAVE) EJEMPLOS

Implementación de los métodos realPart: e imaginaryPart: de la clase complex

realPart: parteReal

"Inicializa la componente real del complejo receptor con parteReal. Retorna el receptor modificado."

realPart ← parteReal.

imaginaryPart: parteImaginaria

"Inicializa la componente imaginaria del complejo receptor con parteImaginaria. Retorna el receptor modificado."

imaginaryPart ← parteImaginaria.

ESTRUCTURAS DE CONTROL

Como en SMALLTALK todo es objeto, las estructuras de control se implementan por medio de mensajes enviados a un objeto.

Las estructuras de control que se desarrollarán son:

⇒ **SELECCIÓN O ALTERNATIVA:** se implementa por medio del envío de mensajes a los objetos

- true
- false.

⇒ **ITERACIÓN O REPETICIÓN:** se implementa por medio del envío de mensajes a objetos de las clases:

- Number.
- BlockContext.
- Collection.

SELECCIÓN

⇒ Existen dos objetos de tipo BOOLEAN en el lenguaje y algunos mensajes enviados a estos objetos permiten la implementación de la SELECCIÓN en Smalltalk.

⇒ Los mensajes que se envían a los objetos true y false llevan como parámetro un bloque sin argumentos. La evaluación de dicho bloque depende del mensaje y del objeto al que se lo envía.

⇒ Los mensajes provistos por Smalltalk permiten el testeo y control condicional

- simple
- compuesto

1 - TESTEO Y CONTROL CONDICIONAL SIMPLE

Sintaxis ifTrue:

objetoBooleano **ifTrue**: bloque

SMALLTALK evalúa una vez el bloque si el mensaje fue enviado al objeto true. En cambio si fue enviado al objeto false no realiza ninguna acción.

Sintaxis ifFalse:

objetoBooleano **ifFalse**: bloque

y el análisis es el inverso.

EJEMPLOS

'cama' < 'casa' **ifTrue**: [↑'cama'].	El resultado de la evaluación de 'cama' < 'casa' es el objeto **true**. A éste se le envía el mensaje **ifTrue** con un bloque que pide que retorne el objeto cadena 'cama'. **Dicho bloque es evaluado**.
'cama' > 'casa' **ifTrue**: [↑'cama'].	El resultado de la evaluación de 'cama' > 'casa' es el objeto **false**. A éste se le envía el mensaje **ifTrue** con un bloque que pide que retorne el objeto cadena 'cama'. **Dicho bloque no es evaluado**.

TESTEO Y CONTROL CONDICIONAL SIMPLE

(continuación)

Métodos ifTrue: e ifFalse:

Clase True

ifTrue: bloque

"Responde el valor del bloque ya que el receptor es verdadero"

↑bloque **value**

ifFalse: bloque

"Responde la alternativa falsa, nil, ya que el receptor es verdadero"

↑nil

Clase False

ifTrue: bloque

"Responde la alternativa falsa, nil, ya que el receptor es falso"

↑nil

ifFalse: bloque

"Responde el valor del bloque ya que el receptor es falso"

↑bloque **value**

2 - TESTEO Y CONTROL CONDICIONAL COMPUESTO

Sintaxis ifTrue:ifFalse:

objetoBooleano **ifTrue**: bloqueVerdadero
ifFalse: bloqueFalso

EJEMPLOS

cadena1 < cadena2
ifTrue: [↑cadena1]
ifFalse: [↑cadena2]

Si la variable cadena1 apunta a un objeto cadena que es menor que el objeto cadena apuntado por cadena2, retorna cadena1, sino retorna cadena2.

numero ← numero < 0
ifTrue: [numero **negated**]
ifFalse: [numero]

Si la variable numero apunta a un objeto numérico que es negativo es retornado le resultado de enviar a ese objeto el mensaje negated y este resultado es asignado nuevamente a la variable numero. Caso contrario no se produce ningún cambio.

TESTEO Y CONTROL CONDICIONAL COMPUESTO

(continuación)

Método ifTrue:ifFalse:

Clase True

ifTrue: bloqueVerdadero **ifFalse:** bloqueFalso

"Responde el valor del bloque Verdadero"

↑bloqueVerdadero **value**

Clase False

ifTrue: bloqueVerdadero **ifFalse:** bloqueFalso

"Responde el valor del bloque Falso"

↑bloqueFalso **value**

3 - MENSAJES QUE REPRESENTAN OPERACIONES LÓGICAS

⇒ Para poder analizar condiciones compuestas, están definidos el mensaje unario not y los mensajes binarios & (AND) y | (OR) en ambas clases true y false.

⇒ Llevan un objeto como parámetro; el objeto retornado depende del objeto al que se le envía el mensaje y del mensaje.

EJEMPLO

```
numero > 0 & (numero < 50)
        ifTrue: [↑'El número está
                    entre 0 y 50']
        ifFalse: [↑'El número está
                    fuera del rango']
```

Una forma de testear si el valor de un número está en un rango es utilizar una condición compuesta. Notar que el objeto alternativo del & es enviado entre paréntesis para no tener problemas con la precedencia.

MENSAJES QUE REPRESENTAN OPERACIONES LÓGICAS (continuación)

Métodos &, | y not

Clase True

not
"Negación lógica. Devuelve falso ya que el receptor es verdadero"
↑false

& objeto
"Conjunción lógica. Devuelve el objeto ya que el receptor es verdadero."
↑objeto

| objeto
"Disjunción lógica. Devuelve verdadero ya que el receptor es verdadero."
↑self

Clase False

not
"Negación lógica. Devuelve verdadero ya que el receptor es falso"
↑true

& objeto
"Conjunción lógica. Devuelve falso ya que el receptor es falso."
↑self

| objeto
"Disjunción lógica. Devuelve el objeto ya que el receptor es falso."
↑objeto

4 - ALGUNOS MENSAJES QUE RETORNAN LOS

OBJETOS TRUE O FALSE

Los siguientes mensajes binarios retornan los objetos true o false. Este comportamiento los hace aptos para formar parte de expresiones que reciben los mensajes que permiten la implementación de la estructura de control SELECCIÓN.

MENSAJE	SIGNIFICADO
==	equivalentes (idénticos)
~~	no equivalentes
=	iguales
~=	no iguales
< <=	menor, menor o igual
> >=	mayor, mayor o igual

Ejercicio 1

Crear el método **ifTrueThen:else:** en la clase Boolean.

Ejercicio 2

Escribir un método que reciba como parámetro un número y retorne una de las siguientes expresiones según corresponda.

El número está entre 0 y 24.

El número está entre 25 y 50.

El número está fuera de rango.

].

ITERACIÓN

La iteración en SMALLTALK es soportada mediante el envío de mensajes a objetos de las clases Number, BlockContext y Collection. La clase Collection tiene varias subclases, como por ejemplo Array.

Los ciclos que veremos son:

⇒ UN EQUIVALENTE AL CICLO DE CANTIDAD DE REPETICIONES CONOCIDAS EN SMALLTALK

- to:do: Y to:by:do:
- timesRepeat:

⇒ UN EQUIVALENTE AL CICLO WHILE EN SMALLTALK

- whileTrue:
- whileFalse:

⇒ COLECCIONES: OTRA FORMA DE REPETICIÓN

- do:
- collect:
- select:
- reject:
- detect:

1 - UN EQUIVALENTE AL CICLO DE CANTIDAD DE REPETICIONES CONOCIDAS EN SMALLTALK

1. a - CICLOS to:do: Y to:by:do:

Un ciclo iterativo simple con un número específico de repeticiones se puede llevar a cabo usando mensajes de la clase Integer.

El método **to:** enteroFinal **do:** unBloque evalúa el argumento unBloque por cada entero en el intervalo [objeto Receptor, objeto enteroFinal]. El argumento del bloque toma sucesivamente los valores del intervalo en cada evaluación del bloque.

Sintaxis:

enteroInicial **to**: enteroFinal **do**: [:parámetro | s1. s2. ... sn]

El método **to:** enteroFinal **by:** incremento **do:** unBloque es una variación del anterior que especifica la cantidad en la cual el argumento del bloque debe incrementarse en cada evaluación. Para incrementos positivos, la evaluación repetitiva termina cuando el índice del ciclo es mayor que el enteroFinal. Para incrementos negativos, el ciclo finaliza cuando el índice del ciclo es menor que enteroFinal.

Sintaxis:

enteroInicial **to**: enteroFinal **by**: incremento **do**: [:parámetro | s1. s2. ... sn]

CICLOS to:do: Y to:by:do:

EJEMPLOS

```
|unArreglo|
unArreglo ← Array new: 10.
1 to: 10 do: [:j |
    unArreglo at: j put: 0.0].
1 to: 10 by: 2 do: [:j |
    unArreglo at: j put: 3.0].
```

Se define una variable temporal llamada unArreglo a la que se le asigna un objeto arreglo de 10 elementos. Luego se inicializan todos sus elementos en 0.0. A continuación se inicializa con 3.0 las posiciones impares de dicho arreglo.

Ejercicio 3:

Crear un método en la clase entero que se denomine **descendiendoHasta:** enteroFinal **hacer:** unBloque que repita la evaluación de dicho bloque para cada entero que está entre el receptor del mensaje y enteroFinal decrementándose en 1.

1 - UN EQUIVALENTE AL CICLO DE CANTIDAD DE REPETICIONES CONOCIDAS EN SMALLTALK

1. b - CICLO timesRepeat:

Otro ciclo iterativo simple con un número específico de repeticiones se puede llevar a cabo usando el mensaje timesRepeat de la clase Integer. Este mensaje lleva como parámetro un bloque sin argumentos que se evaluará tantas veces como lo indique el entero receptor del mismo.

Sintaxis:

entero **timesRepeat**: [s1. s2. ... sn]

EJEMPLO

```
|suma|
suma ← 0.
4 timesRepeat: [ suma← suma + 5 ].
↑suma
```

Obtener el producto de un número usando sólo suma: resolver 4 * 5 como la suma 4 veces de 5.

```
|string|
string : = 'Esta es una cadena de caracteres'.
(string size)  timesRepeat: [...].
...
```

Notar que el objeto al que se le envía el mensaje es un entero, que puede ser el resultado del envío de otro mensaje. Si se quiere repetir algún bloque según la longitud de una string, se debe colocar (string **size**) en vez de un entero determinado. Los paréntesis no son necesarios, fueron colocados para lograr una mayor legibilidad.

2 - UN EQUIVALENTE AL CICLO WHILE EN SMALLTALK

Existen dos mensajes a objetos de la clase BlockContext que proveen un equivalente funcional de los ciclos while de otros lenguajes. Estos mensajes son **whileTrue**: y **whileFalse**:.

Sintaxis:

[bloqueDeTesteo] **whileTrue**: [bloqueEjecutable] (1)

[bloqueDeTesteo] **whileFalse**: [bloqueEjecutable] (2)

(1) Primero es evaluado el bloqueDeTesteo. Si es verdadero, entonces es ejecutado el bloqueEjecutable. Este proceso se repite hasta que el bloqueDeTesteo es evaluado como falso. Para (2) se sigue una lógica similar, pero inversa.

Como se puede ver, éste es un ciclo que se repite de 0 a n veces.

EJEMPLO

```
"Contar las vocales de una string"
|vocales string indice|
vocales ← 0.
indice ← 0.
string ← 'String con varias vocales".
[indice < = string size]
        whileTrue: [
                (string at: indice) isVowel
                        ifTrue: [vocales ← vocales + 1].
                indice ← indice +1].
↑vocales
```

CLASE BlockContext

Notar que la implementación de los dos mensajes se hace en forma recursiva.

whileTrue: unBloque

"Evalúa el argumento unBloque mientras el receptor es verdadero."

```
↑self value
        ifTrue: [ unBloque value.
                self whileTrue: unBloque ]
```

whileFalse: unBloque

"Evalúa el argumento unBloque mientras el receptor es falso."

```
↑self value
        ifFalse: [ unBloque value.
                self whileFalse: unBloque ]
```

3 - COLECCIONES: OTRA FORMA DE REPETICIÓN

La clase Collection provee una variedad de esquemas de iteración. Todos los mensajes llevan como argumento un bloque con un parámetro que es evaluado una vez por cada elemento en el receptor (cada carácter de una string, cada elemento de un arreglo, etc.)

Sintaxis:

objetoColección mensaje: [:parámetro | s1. s2. ... sn]

El mensaje puede ser alguno de los siguientes:

⇒ **do**: se ejecuta el bloque argumento tantas veces como elementos aparezcan en la colección, asignando al parámetro un elemento de la lista en cada ejecución.

⇒ **collect**: para cada uno de los elementos del receptor se evalúa el bloque con ese elemento como parámetro, y devuelve una nueva colección formada por el resultado de esas evaluaciones.

⇒ **select**: crea y retorna una subcolección de los elementos del receptor, que verifican la expresión booleana final del bloque. Es decir que la expresión sn debe ser booleana.

⇒ **reject**: crea y retorna una subcolección de los elementos del receptor que no verifiquen la expresión booleana final del bloque. Es decir que la expresión sn debe ser booleana.

⇒ **detect**: devuelve el primer elemento de la lista que verifique la expresión booleana final del bloque.

EJEMPLOS

|string cuenta|

cuenta ← 0.

string ← 'Orientacion a Objetos'.

string **do**: [:caracter | caracter **isUppercase**

ifTrue: [cuenta ← cuenta + 1]]. **(1)**

string **collect**: [:caracter | caracter **asUppercase**]. **(2)**

string **select**: [:caracter | caracter **isUppercase**]. **(3)**

string **reject**: [:caracter | caracter **isUppercase**]. **(4)**

string **detect**: [:caracter | caracter **isUppercase**]. **(5)**

Los resultados de la evaluación de cada mensaje son los siguientes:

(1) 2, que es la cantidad de caracteres en mayúsculas que hay en la string.

(2) 'ORIENTACION A OBJETOS', que es la string convertida a mayúsculas.

(3) 'OO', que es la subcadena que está en mayúsculas.

(4) 'rientacion a bjetos', que es la subcadena que no está en mayúsculas.

(5) $O, que es el primer elemento (carácter) que está con mayúsculas.

COLECCIONES: OTRA FORMA DE REPETICIÓN

OTROS EJEMPLOS

- Sumar los cinco primeros números primos

  |suma|

  suma ← 0.

  #(2 3 5 7 11) **do**: [:primo | suma ← suma + primo].

  ↑suma

- Obtener el cuadrado de cada uno de los objetos enteros de un arreglo.

  #(1 3 4 9) **collect**: [:i **squared**]

  el resultado es: #(1 9 16 81).

- Extraer los objetos números impares de un arreglo.

  #(1 3 6 7) **select**: [:num | num **odd**]

  el resultado es #(1 3 7).

- Extraer los objetos números pares de un arreglo.

  #(1 3 6 7) **reject**: [:num | num **odd**]

  el resultado es #(6).

- Extraer el primer objeto número impar de un arreglo.

  #(1 3 6 7) **detect**: [:num | num **odd**]

  el resultado es el objeto entero 1.

ANEXO: VARIABLES COMPARTIDAS

DICCIONARIOS POOL (Pool Dictionaries):

Son colecciones de variables (asociaciones nombre/valor) cuyo ámbito es un subconjunto definido de clases en el sistema.

VARIABLES POOL

Son variables compartidas por un subconjunto de clases en el sistema y son almacenadas en diccionarios Pool. No es necesario que exista una relación de herencia entre las clases que forman este subconjunto.

Para que las variables pool se puedan acceder se debe declarar en la definición de la clase el nombre del diccionario Pool que las contienen.

VARIABLES GLOBALES

Son variables compartidas por todos los objetos. Existe un diccionario pool llamado Smalltalk que es accesible globalmente, es decir, es compartido por todas las clases y contiene todas las variables globales.

Con el mensaje **at:put:** se agregan nuevas variables a los diccionarios pool.

EJEMPLOS

1. **Declaración de una variable (#NombreVariable) con un valor (valor) en un diccionario (Diccionario).**

```
Diccionario at: #NombreVariable put: valor.
```

2. **Declaración de una variable (#NombreVariable) con un valor (valor) en el diccionario Smalltalk.**

```
Smalltalk at: #NombreVariable put: valor.
```