Smalltalk

Daniel Delgado, *Estudiante, ITCR,* Wilbert Gonzales, *Estudiante, ITCR,*
Anthony Leandro, *Estudiante, ITCR,* and Bryan Mena, *Estudiante, ITCR*

1. Datos Historicos

Smalltalk fue creado por Alan Kay, fue el primer lenguaje orientado a objetos. El entorno y el lenguaje son muy maduros, se desarrollo entre 1970 y 1980 en un entorno de investigación aislado de todo tipo de comercio, por lo que se convirtió en una estructura muy bien pensada. Ademas, es uno de los lenguajes orientados a objetos "puros". No hay separación entre el lenguaje y el ambiente de desarrollo.

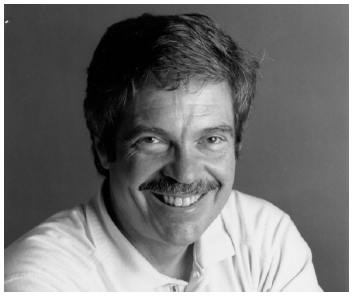

Figura 1. Alan Kay.

2. Objetos y mensajes

Todo en Smalltalk es un objeto, incluyendo:

- Números.
- Strings.
- Windows.
- El compilador.
- Partes del ambiente de desarrollo

Los objetos se comunican por el envío de mensajes, el mensaje puede retornar un objeto o solamente un mensaje indicando que el proceso finalizó. Ejemplos de expresiones validas para Smalltalk:

Expresión	Objeto recibido	Mensaje
3 squared	Número entero 3	squared
'abc' asUppercase	String 'abc'	asUppercase
200 factorial	Número entero 200	factorial

Cada mensaje enviado tiene un receptor, nombre del mensaje y cero o mas argumentos, Smalltalk usa tres diferentes formas de sintaxis.

2.1. Mensajes Unarios

Esta sintaxis es usada cuando no hay argumentos.

Cuadro 1. Mensaje enviado en Smalltalk.

```
miDepa manager name last
```

Cuadro 2. Mensaje enviado en Java.

```
miDepa.manager().name().last()
```

El nombre de la variable es miDepa y los demás son nombres de mensajes.

2.2. Mensajes Binarios.

Este mensaje siempre toma exactamente un argumento y el nombre del mensaje puede ser uno o una secuencia de caracteres especiales de los siguientes:

- +
- *
- <=
- ==

Cuadro 3. Mensaje enviado en Smalltalk.

```
x + y
```

Cuadro 4. Mensaje enviado en Java.

```
x.plus(y)
```

2.3. Mensajes Keyword.

Estos mensajes toman uno o mas argumentos y siempre tendrá al menos un carácter de dos puntos entre ellos. Con ver el nombre del mensaje se puede notar exactamente cuantos argumentos toma el mensaje con solo contar los caracteres de dos puntos.

Cuadro 5. Mensaje enviado en Smalltalk.

```
x addKey: a value: b
```

Cuadro 6. Mensaje enviado en Java.

```
x.addKeyValue(a, b)
```

El nombre del mensaje es 'addKey:value:', el carácter de dos puntos se usa en el envío de estos mensajes ya que es sintacticamente significativo para entender esta sintaxis. Es importante tener en cuenta que el nombre del mensaje no es un token contiguo. Al ser usado el nombre del mensaje se extiende con expresiones de argumentos intermedios.

3. Declaraciones de asignación.

Cuadro 7. Sintaxis general de asignación.

```
var := expr
```

Siempre se tiene el nombre de la variable a la izquierda del operador que asigna. Si "var.ªpunta a un objeto, después de ser asignado "varz .exprvan apuntar al mismo objeto.

4. Conversiones de tipo.

En Smalltalk solo se tiene una forma de convertir información de una representación a otra, es con el envío de mensajes unarios.

Cuadro 8. Ejemplo de conversiones.

```
| d i s |
...
d := i asFloat.
i := d asInteger.
s := i asString.
```

5. Casting.

Smalltalk tiene un tipo de conversión pero no tiene 'casting', ya que no hay verificación de tipos, nunca es necesario evitar la verificación de tipos del compilador.

6. Clases esenciales.

En Smalltalk se tienen las siguientes clases que se deben de conocer para empezar a programar.

6.1. Objeto clase.

Object es la super clase de todas las clases, por lo que todas las clases pueden heredar sus métodos. Los mas comunes son:

6.1.1. printString

Convierte el objeto que recibe a un objeto string. Se usa cuando se necesita representar un objeto en string para mostrar resultados.

Cuadro 9. Código para representar un objeto en string.

```
fuelExample
| price |
price := 100.
Dialog warn:
  'The price is ', price printString
```

Esta definición solo retorna el nombre de la clase, si se crea una nueva clase y se quiere convertir a string mas que el nombre, se debería de cambiar el printString y definir su propio método.

6.1.2. Comparando por igualdad y equivalencia (identidad).

Símbolo	Función	Retorna
=	Verifica si el receptor y argumento son iguales de una forma definida por el programador	true o false
==	Verifica si el receptor y argumento son uno y el mismo objeto (idénticos)	true o false

Cuadro 10. Ejemplo de uso de los comparadores.

```
12 = 12.0        "true"
12 == 12.0       "false"

123 = (122 + 1)    "true"
123 = (122 + 1.0)   "true"

123 == (122 + 1)    "true"
123 == (122 + 1.0) "false"
```

La comparación con = es como que solamente tengan el mismo valor y con == es que sean el mismo objeto.

- ~= significa "no iguales".
- ~~ significa "no identicos".

6.1.3. Proceso de copiado.

Los métodos de copiado hacen copias del objeto recibido y crean un nuevo objeto del mismo tipo que del recibido. La relación entre el objeto recibido y el nuevo objeto depende de como están definidos los métodos. Los 2 principales métodos en este protocolo son:

- copy: Primero crea una copia superficial del objeto recibido y luego envía un mensaje postCopy al objeto recibido para realizar cualquier otra operación de copia que se desee.
- deepCopy: Copia el objeto recibido y luego de manera recursiva pasa sobre todos los objetos referidos y también los copia.

6.2. Clases numéricas.

Estas clases son las mas importantes de utilizar en Smalltalk:

- Enteros.
- Números de punto flotante.
- Fracciones.
- Número fijo de dígitos decimales.
- Números complejos.
- Metanumeros.

Ejemplo de operaciones aritméticas con números:

Cuadro 11. Operaciones a números.

```
3 + 7 / 3
"Mensaje del protocolo aritmetico."
(3 + 7 / 3) asFloat
"asFloat: del protocolo de conversiones."
(3 + 7 / 3) asFixedPoint: 2
"asFixedPoint: del protocolo de
conversiones."
Float pi asRational
"Tambien del protocolo de conversiones."
15 log
"log: es funcion matematica."
0.3 sin
"sin: es funcion matematica."
1000 factorial
"Protocolo de factorizacion y division."
37 raisedTo: 22
"raisedTo: es funcion matematica."
16rF8 + 2r00001000
"constante se puede obtener
en base hex, binaria y otras."
```

6.3. Strings

Un string es una colección de caracteres indexada que empieza desde el indice 1. Las operaciones a los strings como la concatenación se da por medio de mensajes.

Cuadro 12. Ejemplo de mensajes con strings.

```
'abcdefg' findString: 'de' startingAt: 1
"findString: retorna el indice
o 0 si no lo encuentra"

'abcdefg' size
"size: retorna el tamano del string"

'abcdefg' copyFrom:2 to:4
"Copia el caracter del indice 2 al 4.
```

6.4. Caracteres.

Cada elemento individual de un string son instancias de la clase Carácter. Un carácter se puede crear por la conversión del código numérico, extraer de un string o crear una instancia con el mensaje.

Cuadro 13. Ejemplo de uso de caracteres.

```
Character value:8
"Caracter del codigo ASCII"
80 asCharacter
"Caracter del codigo ASCII"
'hola' at:2
"El segundo elemento del string"
Character cr
"Instancia"
Character esc
```

6.5. Símbolos.

Los símbolos son similares a los strings, pero estas instancias son únicas y los strings no lo son. Lo que quiere decir que al crear un nuevo simbolo Smalltalk primero busca que exista otro símbolo con el mismo contenido y devuelve ese objeto existente. Los símbolos literales se escriben como strings pero con un # antes. Para símbolos alfanuméricos y para selectores binarios se pueden omitir las comillas simples.

Cuadro 14. Ejemplo de simbolos

```
#'abc' = #'abc'
#'abc' == #'abc'
"retorna true"
"ya que los simbolos son iguales"
```

6.6. Dialogs.

Contiene muchos mensajes de utilidad para obtener datos como string del usuario.

Cuadro 15. Ejemplo de uso y tipos de dialogs.

```
Dialog request: 'What is your age?'
"Retorna string"
Dialog request: 'What is your age?'
    initialAnswer: '20'
"Retorna string, tiene por defecto 20"
(Dialog request: 'What is your age?'
    initialAnswer: '20') asNumber
"Retorna un numero"
Dialog confirm: 'Are you sure you
    want to delete the file?'
"Retorna true o false"
Dialog warn: 'This is a warning'
"Retorna nil"
Dialog information: 'This is some
information'
"Retorna nil"
```

6.7. Text y ComposedTextView.

La clase string no tiene información como color, tipo de letra, etc. Por lo que se imprime con las propiedades default y si se quiere manipular estas características se debe de convertir el string al objeto Text.

Cuadro 16. Ejemplo de uso.

```
ComposedTextView open:
        'abc' asText allBold asValue
```

Esto abre una ventana con el texto especificado en negrita.

6.8. Boolean.

La clase boolean es abstracta y las clases True y False son subclases, cada una tiene su propia instancia. Principalmente es usado para controlar el flujo de ejecución, como la ejecución condicional de un bloque de instrucciones y la repetición condicional.

Cuadro 17. Ejemplo de uso.

```
(4 < 5)
"true"
 (4 < 5) ifTrue:
  [Transcript clear; show: '4 es menor que 5']
ifFalse:
  [Transcript clear; show: '4 es mayor que 5']
```

Los constructores de paréntesis cuadrados constituyen bloques, la evaluación de las declaraciones dentro de bloques se retrasan hasta que la definición del mensaje lo solicite explícitamente.

6.9. Blocks.

Un bloque es la ejecución retardada de una secuencia de cero o mas instrucciones, las expresiones dentro del bloque se evalúan solo si el programa lo solicita explícitamente. Son muy importantes y el uso mas común es para la iteración, ya que se da la evaluación repetida de una secuencia de enunciados mientras se cumple una condición.

Cuadro 18. Ejemplo de uso.

```
| count |
count := 0.
[count < 100] whileTrue: [count := count + 1].
"Repite la iteracion hasta que se evalue
como falso el primer bloque."
Transcript clear; show: count printString
```

6.10. Colecciones.

Las clases definen varios tipos de colecciones de objetos y esta es una de las grandes ventajas de Smalltalk. Incluye colecciones con elementos indexados y colecciones desordenadas, cuyos elementos no son accesados por un indice. La mayoría de colecciones son dinámicas ya que su tamaño puede ser cambiado en tiempo de ejecución pero existen un par que tienen tamaño fijo. Las colecciones que son indexadas empiezan con el indice 1. Ademas, la mayoría de colecciones aceptan cualquier objeto pero un par solo aceptan cierto tipo de objetos.

6.10.1. Collection.

Es una super clase abstracta de todas las colecciones. No tiene instancias ya que su propósito es definir todo lo que comparten la mayoría de las colecciones.

6.10.2. Array.

Es indexado y el primer indice es 1, los elementos no pueden ser agregados o eliminados, solo sus valores pueden cambiar.

Cuadro 19. Array creado como literal.

```
#(1 2 3 $a $b $c 'abc' 3.14 #symbol true)
```

Cuadro 20. Array vacio con un tamaño predefinido.

```
|a|

a := Array new:100.
a at:50 put:'fifty'.
a inspect.
```

Cuadro 21. Los elementos del array pueden ser arrays.

```
#(
(1 'one')
(2 'two')
(3 'three')
)
```

6.10.3. OrderedCollection.

Similar al array pero es de tamaño variable. Los usos son algo diferentes.

Cuadro 22. Uso de OrderedCollection.

```
|o|

o := OrderedCollection new.
o add:'one'.
o add:'two'.
o add:'three'.
o inspect.
```

6.10.4. SortedCollection.

Es una subclase de OrderedCollection que ordena sus elementos usando un bloque de clasificación. Por defecto usa sort block.

Cuadro 23. Bloque para ordenar elementos.

```
[:elemento1 :elemento2 | elemento1 < elemento2]
```

6.10.5. Set.

Es una colección desordenada, sin indice y elimina automáticamente los elementos duplicados.

Cuadro 24. Ejemplo para convertir a un Set.

```
#(1 1 2 3 3 2.0) asSet
```

6.10.6. IdentitySet.

Es como el Set, pero usa identidad en lugar de igualdad para comprobar la duplicación, la verificación de identidad es mas rápida que la de igualdad.

Cuadro 25. Ejemplo para convertir a IdentitySet.

```
#(1 1 1 2 2 2 3 3 3 2.0) asIdentitySet
```

6.10.7. *Dictionary.*

Es un conjunto de asociaciones, una asociación es un par con clave-valor y se crea con el mensaje binario $- >$.

Cuadro 26. Ejemplo para agregar a un diccionario.

```
Dictionary new
add: 'CRT' -> 'Cathode Ray Tube';
add: 'SSI' -> 'Small Scale Integration';
yourself
```

Cuadro 27. Ejemplo para actualizar diccionario.

```
Dictionary new
add: 'key1' -> 'old value';
add: 'key1' -> 'new value';
yourself
```

6.10.8. *IdentityDictionary.*

Es similar al diccionario, pero usa la identidad para probar la duplicación de elementos. ademas es mas eficiente que el diccionario ya que las pruebas de identidad son mucho mas rápidas que las pruebas de igualdad.

Cuadro 28. Ejemplo con 2 entradas en la misma llave.

```
IdentityDictionary new
add: 'key1' -> 'old value';
add: 'key1' -> 'new value';
yourself
```

6.10.9. *Bag.*

Es como el Set, pero la bolsa sabe cuantas apariciones de un elemento se presentan.

Cuadro 29. Ejemplo de Bag.

```
#(1 1 1 2 2 2 3 3 3) asBag
```

6.10.10. *Interval.*

Es una representación compacta de una progresión aritmética, es una secuencia de números igualmente espaciados.

Cuadro 30. Representa una secuencia 3, 4, 5, 6, 7, 8, 9.

```
Interval from: 3 to: 9
```

7. DECLARACIONES DE FLUJO DE CONTROL.

Cada una de las declaraciones de flujo de control se hace usando un mensaje. Ejemplo: La instrucción ïfüsa el mensaje ïfTrue: ifFalse:”.

7.1. La declaración *If*

La instrucción if inicia con la expresión booleana. Los paréntesis cuadrados se requieren, incluso si solo hay una declaración. Ambos argumentos para el método ’ifTrue: ifFalse:’ deben de ser bloques.

Cuadro 31. Sintaxis de un If

```
(x < y) ifTrue: [
max := y.
i := j
] ifFalse: [
max := x.
i := k
]
```

Hay solo 3 tipos diferentes de mensajes:

- ifTrue:ifFalse:
- ifTrue:
- ifFalse:

7.2. La declaración *While*

La expresión debe de estar entre paréntesis, ya que debe de evaluarse cada vez que hace el ciclo

Cuadro 32. Ejemplo de un ciclo ”while”.

```
[i < 100] whileTrue: [
sum := sum + i.
i := i + 1
]
```

7.3. La declaración de *For*

Cuadro 33. Ejemplo de un ciclo ”for”.

```
1 to: n do: [ :i |
sum := sum + i.
]
```

El cuerpo del ciclo se ejecuta iterativamente con valores incrementados en 1 de ’i’, en el rango de 1 a n. La variable ’i’ es local al cuerpo del ciclo y no puede ser accesible desde otra parte.

7.4. La declaración de *Return*

Se usa el carácter ˆpara retornar un valor

Cuadro 34. Comparación de return

```
^ i+6          "Smalltalk"

return i+6;    //Java
```

7.5. Comentarios en el código

Los comentarios se encierran en comillas dobles. Los comentarios son mas fáciles de leer junto al código. Ademas, al hacer comentarios se ven integrados al código y es mas fácil entenderlo que es un documento aparte, aunque los comentarios están integrados al código no se destacan tanto en Smalltalk.

7.6. Comparación de nombres predefinidos con Java.

Son muy pequeñas las diferencias en la sintaxis, también hay varios nombres predefinidos que son identificadores de objetos específicos.

Smalltalk	Java
1234	1234
11,34	11,34
$a	$'a'$
$'hello'$	$"hello"$
nil	$null$
$true$	$true$
$false$	$false$
$self$	$this$
$super$	$super$

7.7. Variables de clases.

Es una variable que puede ser accesada globalmente en la clase. Solo hay una copia de cada variable y es compartida por todas las instancias, se puede acceder a una variable de clase desde cualquier parte del código mientras se este dentro de la clase, sin necesidad de una referencia a alguna instancia en especifico. Por ejemplo, se tiene la clase Persona, se necesita un contador de las instancias de Persona, entonces solo se agrega a la clase Persona una variable llamada PersonaCount y cada ves que se cree una instancia, se aumenta el contador en uno.

Cuadro 35. Ejemplo de contador de la clase Persona.

```
PersonaCount := PersonaCount + 1
```

7.8. Métodos de clases.

Método que es parte de una clase, pero no es invocado en una instancia particular, ademas se pueden invocar antes de que se hayan creado instancias. Por ejemplo, se tiene la clase Persona con una variable que es el total de instancias de la clase Persona, se crea un método de la clase llamado 'printStatics' para imprimir el valor de todas las variables de la clase. Para invocar un método de una clase, un mensaje es enviado a la misma clase.

Cuadro 36. Ejemplo de metodo de la clase Persona

```
Person printStatistics
```

7.9. Creación de objetos.

Toda clase se representa por un objeto. Ejemplo: La clase Estudiante es representada por el objeto Estudiante. Al invocar un método de la clase, se invoca un método de la instancia por el envío de mensajes a una instancia de la clase.

Cuadro 37. Ejemplo para crear una instancia de la clase Persona.

```
p := Persona new
```

Normalmente, el método 'new' es heredado de una implementación en la clase 'Object', creara un nuevo objeto y lo retornara. Se puede sobrescribir el método 'new', por ejemplo para incrementar la variable contador de la clase Persona. Así, le damos diferentes valores iniciales ya que las variables y campos, son inicializados en 'nil'.

7.10. Constructores.

No existen en Smalltalk, pero brinda la misma funcionalidad sin nueva semántica o sintaxis. En Java o C++ los constructores inicializan nuevas instancias y también agregan valores iniciales. En Smalltalk también se modifican algunas variables estáticas como 'PersonaCount' dentro del constructor, por ejemplo se puede crear un método de clase que tome dos parámetros.

Cuadro 38. Ejemplo para crear una instancia de la clase Persona.

```
p := Person newNombre: 'John' edad: 15
```

En este ejemplo el método de la clase hereda del método 'new' para crear el objeto y luego modifica unos campos y también actualiza la variable de la clase.

7.11. Operadores matemáticos.

Operador	Significado
$+, -$	Suma, Resta
$*, /$	Multiplicación, División
$bitAnd, bitOr, bitXor$	Operadores bitwise
$bitInvert$	Invertir bitwise
$bitShift$	Operadores bitwise
$raisedTo$	Exponente
log : 10	Logaritmo base 10
ln	Logaritmo natural
rem	Modulo con negativos
$\backslash\backslash$	Modulo valor abs
$-$	Negacion
$1000{,}0, 1E3$	Sintaxis punto flotante
$2r1$	Sintaxis enteros con base 2
100	Sintaxis de numero entero
$sqrt$	Raíz cuadrada
$raisedTo$	Exponente
exp, abs	Exponente e, valor absoluto
sin, cos, tan	Trigonometria básica
$arcSin, arcCos, arcTan$	Trigonometria inversa
$rounded, floor$	Redondear
$Float, Double$	Nombre de tipos de punto flotante
$Fraction, FixedPoint$	Nombre de tipos de punto flotante
$Integer, SmallInteger$	Nombre de tipos enteros
$LargeInteger$	Nombre de tipos enteros

7.12. Operadores Booleanos

Operador	Significado
and	and lógico (short circuit)
or	or lógico (short circuit)
not	Negación lógica
false	Valor falso
true	Valor verdadero
Boolean	Nombre del tipo
|	or lógico (evalúa ambos argumentos)
&	and lógico (evalúa ambos argumentos)

8. Herencia.

Las clases están relacionadas, la clase Object es la única que no tiene una superclase, el resto de clases tienen exactamente una superclase y tiene todas las variables de instancia de su superclase y comprende todos sus mensajes. Si A es superclase de B y B es la superclase de C, C hereda de B incluyendo todo lo que B hereda de A, por lo que C hereda todo lo de B y A. Es transitiva. Entonces todas las clases heredan todo el comportamiento de la clase Object.

Cuadro 39. Ejemplo para conocer la superclase de una clase.

```
Fraction superclass
```

9. Self y Super.

'Self' se refiere al receptor en si, es significado es idéntico al 'this' de Java. 'Super' y 'self' se refieren al mismo objeto, el receptor del método, la diferencia es que 'super' busca desde la superclase del método que usa 'super'.

Cuadro 40. Ejemplo del "Self".

```
self computeTaxes.
phoneBook at: self put: i
```

10. Accesando campos de objetos.

En Smalltalk, los campos de un objeto no se pueden acceder desde fuera de la clase. Ademas, todos los campos son privados, entonces todos los accesos externos deben de darse por medio de métodos, que se conocen como métodos de acceso.

Cuadro 41. Ejemplo de método de acceso al campo "name"de una clase.

```
name
        "Retorna."
        ^ name

name: str
        "Actualiza."
        name := str
```

11. Características

- Interacción entre objetos mediante el envío de mensajes.
- Se da herencia simple y con raíz común.
- Garbage collector.
- El código fuente es parte del entorno, por lo que es disponible para su extención, modificación y estudio.
- Es fuertemente tipado, es decir un objeto no puede ejecutar un mensaje que no fue programado.
- Es dinamicamente tipado, es decir al evaluar un mensaje, encuentra la definición en tiempo de ejecución.

12. Ventajas

- Las reglas del lenguaje son simples.
- El ambiente de desarrollo es muy poderoso.
- El código que escribe un programador con experiencia en Smalltalk es legible y muy compacto.
- Es un lenguaje muy flexible y las aplicaciones son muy extensibles.
- No se necesita el çasting"de tipos.

13. Desventajas

- La sintaxis es distinta a otros lenguajes, ya que sigue el paradigma de envio de mensajes.
- Algunos entornos no incorporan manejo de excepciones.
- No es estandarizado, es decir se presentan muchas diferencias entre versiones.

Referencias

[1] T. Budd. 1987. *A Little Smalltalk.*Recuperado de http://sdmeta.gforge.inria.fr/FreeBooks/ LittleSmalltalk/A-LittleSmalltalk.pdf

[2] W. Lalonde. 1994. *Discovering Smalltalk.* Object Technology Series. Recuperado de https://books.google.co.cr/books/about/Discovering_Smalltalk.html?id=t4RQAAAAMAAJ&redir_esc=y

[3] S. Lewis. 1995. *The Art and Science of Smalltalk.* Textbook Binding. Recuperado de http://sdmeta.gforge.inria.fr/FreeBooks/Art/ artAdded174186187Final.pdf

[4] J. Pletzke. 1985. *Advanced Smalltalk.* Computer Publishing. John Wiley & Sons, Inc.

[5] D, Gomez. 2006. *Libro de programación con Smalltalk.* Recuperado de http://diegogomezdeck.blogspot.com/2006/02/libro-programando-con-smalltalk.html