Lenguajes de Programación

Dr. Carlos A. Coello Coello
Departamento de Computación
CINVESTAV-IPN
ccoello@cs.cinvestav.mx

Iteración

- Los ciclos con la decisión al inicio pueden formarse en Smalltalk usando la palabra clave "**method whileTrue**:", la cual es proporcionada por la clase "Block".

- Esta palabra clave se usa para enviar el bloque a ser controlado por un segundo bloque que contiene la condición del ciclo.

Iteración

- Este método está definido para todos los bloques que regresen objetos Booleanos.
- El método "**whileTrue:**" está definido para enviar "**value**" al objeto que contenga el método (ya sea "true" o "false"), causando en consecuencia que se ejecute el bloque que se le pasó como parámetro.

Iteración

Ejemplo:

```
count ← 0.
sum ← 0.
[count <= 20] “El bloque con la condicion
del ciclo”
    whileTrue: [sum ← sum + count.
                count ← count + 1]  ”El
cuerpo del ciclo”
```

Iteración

- Otra estructura de control común para ciclos es la simple repetición mediante un contador.
- Para esto, existe un método para los enteros llamado "**timesRepeat:**".
- Cuando se envía "**timesRepeat:**" a un entero con un bloque como parámetros, dicho bloque se ejecuta el número indicado de veces.

Iteración

- Ejemplo:

```
xCube ← 1.
3 timesRepeat: [xCube ← xCube * x]
```

- Esto calcula el cubo de “x” mediante un proceso bastante tedioso.

Iteración

- Pueden construirse estructuras de control para ciclos similares a las de Algol-68 usando algunos de los métodos para enteros que proporciona Smalltalk.

- Los dos métodos más útiles son: "**to:do:**" y "**to:by:do**".

Iteración

- Ejemplo:

```
1 to: 5 do: [sum ← sum + x]
```

- Este bloque es ejecutado 5 veces.
- Los valores internos producidos y regresados por el objeto “1” son: 1, 2, 3, 4 y 5.

Iteración

- Un ejemplo del segundo método es:

```
2 to: 10 by: 2 do: [:even | sum ← sum +
                          even]
```

- Este mensaje hace que el bloque se ejecute 5 veces, pero en este caso los valores internos producidos son: 2, 4, 6, 8 y 10.

Selección

- Se proporciona el método “ifTrue:ifFalse:” para los objetos “true” y “false”.
- Los dos argumentos del mensaje “ifTrue:ifFalse:” representan las cláusulas “then” y “else” del constructor de selección.

Selección

- Este mensaje se envía a una expresión Booleana.
- Si la expresión se evalúa a cierto (true), entonces el mensaje se envía a “true”.
- En este caso, el método “**ifTrue:ifFalse:**” envía “value” a su primer argumento e ignora el segundo.
- Si se evalúa a falso, entonces ocurre lo opuesto.

Selección

- Ejemplo:

```
total = 0
    ifTrue: [average ← 0]
    ifFalse: [average ← sum // total]
```

Selección

- En este caso, la expresión Booleana “total=0” hace que el mensaje “=0” se envíe al objeto “total”.
- Esto regresa ya sea “true” o “false”.
- El objeto resultante es usado después como el receptor del mensaje, el cual se envía al método “**ifTrue:ifFalse**”.

Selección

- Los dos parámetros de este método son los bloques “then” y “else”, respectivamente, de los cuales sólo uno es ejecutado.

- El operador // especifica que se desea realizar una división entera.

Métodos

- Un método de una clase define las operaciones que una instancia de la clase ejecutará cuando se reciba un mensaje correspondiente a ese método.

- En cierto sentido, los métodos son como las definiciones de funciones, incluyendo el uso de parámetros y la capacidad de regresar valores.

Métodos

- La forma sintáctica general de un método en Smalltalk es:

```
patrón_del_mensaje [ | variables temporales | ]
sentencias
```

- donde los corchetes son meta-símbolos que indican que lo que se encuentra dentro de ellos es opcional.

Métodos

- Dado que Smalltalk no tiene declaraciones de tipo, las variables temporales, cuando están presentes, sólo necesitan ser nombradas en una lista.
- Las variables temporales existen sólo durante la ejecución del método en el cual están listadas.
- No hay signos de puntuación al final de un método.

Métodos

- El mensaje del patrón corresponde a las sentencias procedurales de lenguajes imperativos tales como Pascal.

- Los patrones de mensajes, los cuales son prototipos para los mensajes, pueden estar en una de dos formas básicas.

Métodos

- Para los mensajes unarios o binarios, sólo se incluye el nombre del método.
- Para los mensajes de palabra clave, se incluyen las palabras clave y los nombres de los parámetros del patrón de mensaje.

Métodos

- Para indicar que un método regresará un valor, se utiliza la flecha hacia arriba (^).
- En muchos casos, esta es la última expresión que aparece en el método.
- Si no se especifica un valor de retorno en un método, el objeto receptor mismo es el valor de retorno.

Clases

- Todos los objetos de Smalltalk son instancias de clases.

- Una clase tiene cuatro partes:

1) Un nombre (de la clase).

2) El nombre de la superclase, el cual especifica la posición de la nueva clase en la jerarquía de clases del sistema.

Clases

3) Una declaración de las variables locales, llamadas variables instanciadas.

Estas declaraciones estarán disponibles a las instancias de la clase.

Clases

- 4) Los métodos que definen cómo responderán las instancias de la clase a los mensajes.

- (Recordemos que un objeto hereda también el método de todas las clases que son sus ancestros).

Clases

- Los mensajes a un objeto normalmente hacen que se busque un método correspondiente en la clase a la que el objeto pertenece.

- Si la búsqueda falla, se continúa ésta en la superclase de esa clase y así sucesivamente, hasta llegar a la clase del sistema, llamada "**Object**", la cual no tiene superclase.

Clases

- Si no se encuentra un método en ninguna parte de esa cadena, entonces ocurre un error.
- Es importante recordar que este método de búsqueda es dinámico.

Clases

- Ejemplo de una definición de clase:

"Smalltalk Example Program"
"The following is a class definition, instantiations of which can draw equilateral polygons of any number of sides"

class name	Polygon
superclass	Object

Clases

instance variable names ourPen
numSides
sideLength

```
"Class methods"
 "Create an instance"
new
 ^ super new getPen
```

Clases

```
"Get a pen for drawing polygons"
getPen
    ourPen ← Pen new

"Instance methods"
"Draw a polygon"
draw
    numSides timesRepeat: [ourPen go: sideLength;
                                turn: 360 // numSides]
```

Clases

```
"Set length of sides"
length: len
    sideLength  len

"Set number of sides"
 sides: num
    numSides  num
```

Clases

- Smalltalk sigue el **Principio de Abstracción** al agrupar los objetos que tienen comportamiento y propiedades similares bajo la misma clase.

- Nótese que simultáneamente, los particulares que distinguen un objeto de otro se omiten.

Clases

- Este enfoque resulta ideal para simulación y es una muestra de la fuerte influencia de Simula-67 en Smalltalk.
- La memoria privada de una instancia de una clase contiene sus variables instanciadas.

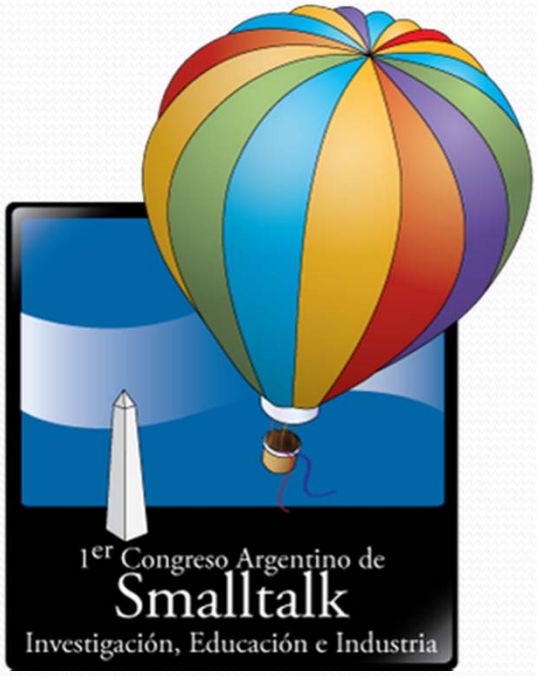
Clases

- Las variables instanciadas no son visibles a otros objetos.
- Cada variable instanciada se refiere a un objeto, al que se denomina “valor”.
- Los valores de todas las instancias de una cierta variable representan (en su conjunto) el estado actual de esa instancia.

Clases

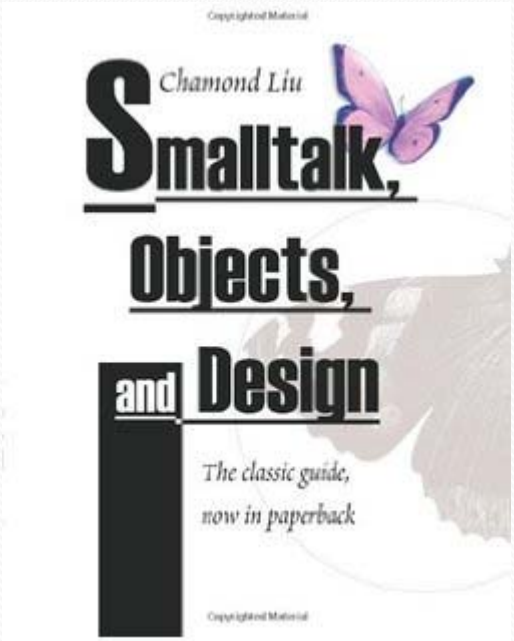
- Las variables instanciadas pueden ser “nombradas” o “indizadas”.
- Las variables nombradas corresponden a apuntadores a tipos distintos de los arreglos en un lenguaje imperativo.
- Las variables indizadas se acceden, no mediante su nombre, sino mediante los mensajes que usan parámetros enteros.

Clases



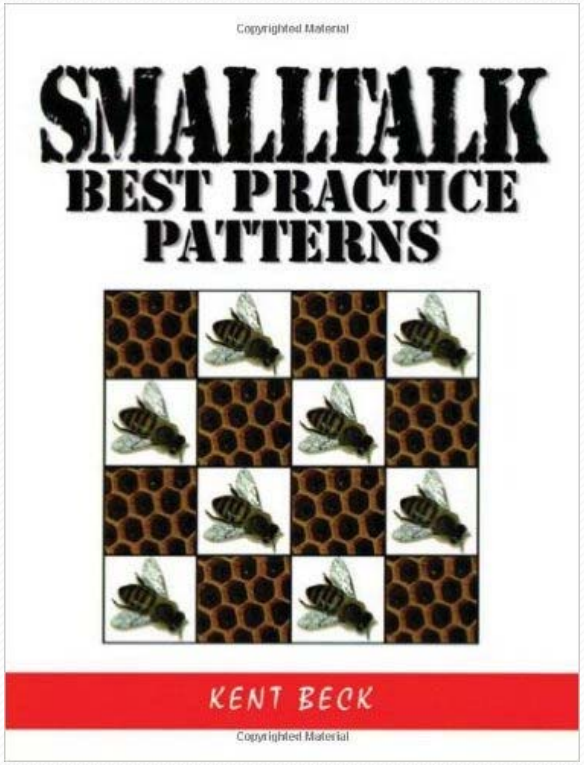
- La mayor parte de las variables indizadas se usan de tal forma que corresponden a arreglos en lenguajes imperativos convencionales, aunque la indización misma se realiza a través del paso de mensajes.

Clases



- El parámetro entero en un mensaje se usa para referenciar una variable indizada instanciada y corresponde al subíndice de un arreglo en un lenguaje imperativo convencional.

Clases



- Las instancias de una clase se crean enviando el mensaje "**new**" a la clase en una asignación, la cual hace que la variable a su izquierda haga referencia al nuevo objeto recién creado.

Clases

- Por ejemplo:

```
ourPen ← Pen new
```

- crea una instancia de la clase “Pen” (enviando el mensaje “**new**” a la clase “Pen”).

Clases

- Esto también hace que la variable “ourpen” haga referencia a esta nueva instancia de la clase “Pen”.
- Este ejemplo demuestra que pueden enviarse mensajes tanto a las clases como a los objetos.

Chequeo de Tipos

- Las variables de Smalltalk no tienen tipo; cualquier nombre puede asociarse con cualquier objeto.
- El único chequeo de tipos que se efectúa ocurre cuando dinámicamente cuando un mensaje se envía a un objeto.

Chequeo de Tipos

- Si el objeto es una de sus clases previas (ancestro), entonces tiene un método para el mensaje, el mensaje es legal y el objeto reacciona a él.

- Si no hay método que corresponda a un mensaje, ya sea en el objeto en una de sus clases ancestrales, entonces se produce un error en tiempo de ejecución.

Chequeo de Tipos

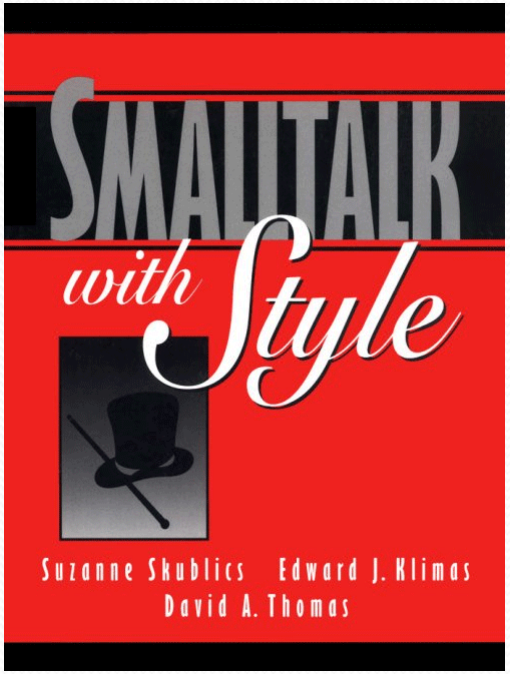
- Este es un concepto de chequeo de tipos significativamente diferente del utilizado en los lenguajes imperativos.

- El chequeo de tipos en Smalltalk tiene como objetivo simplemente asegurar que un mensaje corresponde a un cierto método.

Chequeo de Tipos

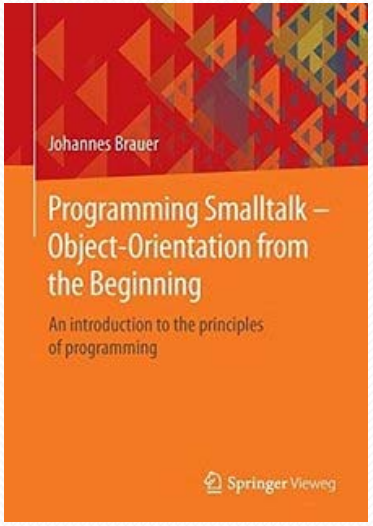
- Advierta que el chequeo dinámico de tipos no implica en este caso un chequeo débil de tipos.
- Smalltalk tiene un chequeo fuerte de tipos, a pesar de que éste es dinámico (en vez del chequeo estático de lenguajes como Pascal y Ada).

Chequeo de Tipos



- Una ventaja del chequeo dinámico de tipos es su flexibilidad, ya que podemos hacer el mismo trabajo que con los paquetes genéricos de Ada, pero sin recurrir a la inherente complejidad de esa técnica.

Chequeo de Tipos



- Puesto que Smalltalk es un lenguaje interpretado y no compilado, el activar mensajes en tiempo de ejecución es algo aceptable y la motivación para detectar tan temprano como sea posible un error que se aplica a los lenguajes compilados no tiene razón de ser en este caso.

Chequeo de Tipos

- Asimismo, no hay violaciones al **Principio de Seguridad**, porque Smalltalk sigue checando si un objeto tiene un método que responda a un cierto mensaje y, de no encontrarlo, activa un error.

- De tal forma, se impide que el programa se colapse debido a violaciones al sistema de tipos.

Herencia

- La herencia es naturalmente un concepto central en Smalltalk.
- El sistema de Smalltalk incluye una enorme jerarquía de clases.
- La programación en Smalltalk consiste, en gran medida en crear subclases de las clases existentes.

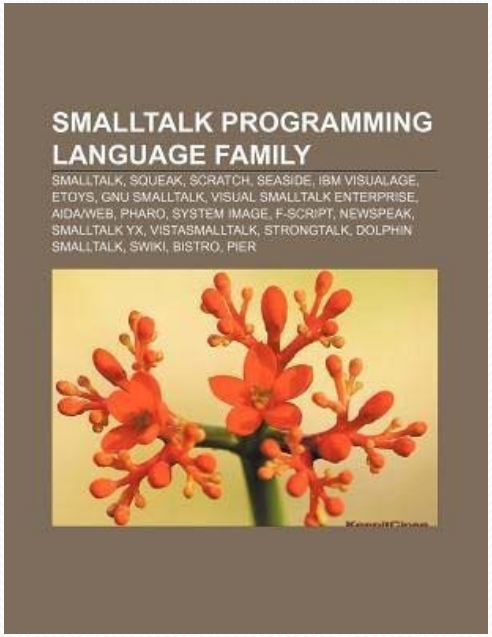
Herencia

- Es precisamente este reuso potencial de código existente lo que resulta más interesante y valioso sobre el mecanismo de herencia.
- De hecho, la herencia es quizás la contribución más importante de los lenguajes orientados a objetos.

Herencia

- Las subclases pueden construirse basadas en el comportamiento de sus superclases, pero también pueden modificarlo.
- Recordemos que en los lenguajes estructurados, una re-declaración de una variable anula la declaración previa.

Herencia



- Lo mismo ocurre en Smalltalk con los métodos: la definición de un método en una subclase anula cualquier definición que pueda existir de ese mismo método en sus superclases.

Herencia

- Un objeto puede enviarse un mensaje a sí mismo porque es un miembro de su superclase.
- Como puede verse, Smalltalk es un sistema muy extensible, en el cual el reuso de software tiene un significado real.

Herencia

- Un problema con las jerarquías definidas en Smalltalk es que su organización impide el uso de una clasificación ortogonal.

- En la vida real, frecuentemente nos encontramos con objetos que deben ser clasificados de varias formas diferentes.

Herencia

- Por ejemplo, un biólogo podría clasificar a los mamíferos como primates, roedores, rumiantes, etc.
- Alguien interesado en los usos de los mamíferos podría clasificarlos como mascotas, bestias de carga, fuentes de comida, etc.
- Finalmente, un zoológico podría clasificarlos como Norteamericanos, Sudamericanos, Africanos, etc.

Herencia

- Estas son 3 clasificaciones ortogonales: cada una de las clases corta a través de las otras en ángulo recto.

- En Smalltalk, el programador está forzado a violar ya sea el **Principio de Seguridad** o el **Principio de Abstracción**, a fin de modelar una clasificación ortogonal.

Herencia

- Es decir, tenemos que repetir atributos de algunos objetos en otras clases (violando el **Principio de Abstracción**) o tenemos que efectuar una clasificación anidada de clases (violando el **Principio de Seguridad**).

- La fuente del problema es que un objeto puede pertenecer a sólo una clase a la vez.

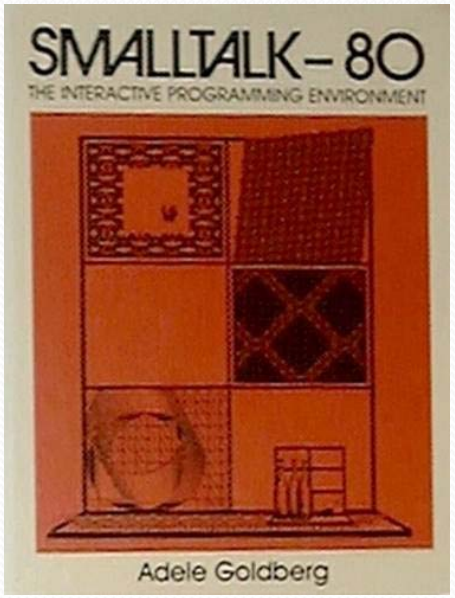
Herencia

- Cuando se define una clase, puede especificarse que será una subclase inmediata de exactamente una clase (y sólo una).
- Una solución posible a este problema que sigue siendo tema de investigación es permitir la herencia múltiple.

Implementación: Clases y Objetos

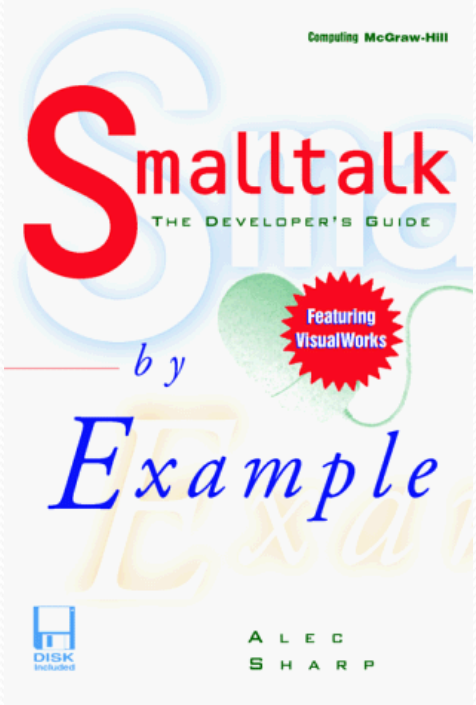
- Una de las cosas más interesantes de Smalltalk es que la mayor parte de su sistema está escrito en Smalltalk.
- Esto incluye el compilador, el de-compilador, el depurador, los editores y el sistema de archivos.
- En total, esto constituye alrededor del 97% del código de Smalltalk.

Implementación: Clases y Objetos



- Una de las razones principales por las cuales Smalltalk puede programarse en Smalltalk es que la mayor parte de las estructuras de datos de la implementación, tales como los registros de activación del lenguaje, son objetos de Smalltalk.

Implementación: Clases y Objetos



- Esto significa que pueden ser manipulados mediante programas en Smalltalk y que tienen las propiedades de los objetos de Smalltalk (por ejemplo, se auto-despliegan).

Implementación: Clases y Objetos

- La parte de Smalltalk que no es portátil es llamada "Máquina Virtual de Smalltalk-80"; su tamaño va de 6 a 12K en ensamblador.

- Los diseñadores de Smalltalk aseguran que toma alrededor de un año producir una versión completamente depurada de una "Máquina Virtual de Smalltalk".

- Esto hace de Smalltalk un excelente ejemplo del **Principio de Portabilidad**.

Implementación: Clases y Objetos

- La Máquina Virtual de Smalltalk tiene 3 componentes principales:

- 1) **Manejador de Objetos:** Es el manejador de tipos de datos abstractos para los objetos.

- Encapsula la representación de los objetos y la organización de la memoria.

Implementación: Clases y Objetos

- Las únicas operaciones que otros módulos pueden realizar sobre los objetos son aquellas proporcionadas por el Manejador de Almacenamiento (*Storage Manager*):

  1) Tomar la clase de un objeto.
  2) Tomar y almacenar los campos de los objetos.
  3) Crear nuevos objetos.

Implementación: Clases y Objetos

- Puesto que el Manejador de Almacenamiento tiene que ser capaz de crear nuevos objetos, debe poder obtener espacio libre donde colocarlos, también.

- Por lo tanto, otra responsabilidad del Manejador de Almacenamiento es colectar y manejar el espacio libre de memoria.

Implementación: Clases y Objetos

- Para este propósito, usa una estrategia de conteo de referencias con extensiones para poder manejar estructuras cíclicas.

2) **Intérprete:** Es el corazón del sistema de Smalltalk.

Implementación: Clases y Objetos

- Aunque sería posible interpretar directamente las formas escritas en Smalltalk, es más eficiente si el intérprete opera sobre una forma intermedia del programa.

- El intérprete es esencialmente el manejador de tipos de datos abstractos para los métodos.

Implementación: Clases y Objetos

- 3) **<u>Subrutinas Primitivas</u>:** Son simplemente una colección de los métodos que, por razones de desempeño, se implementan en lenguaje máquina en vez de hacerse en Smalltalk.

- Aquí se incluyen las funciones básicas de entrada/salida, la aritmética con enteros, el indizamiento de los objetos que lo requieren (p.ej., los arreglos) y las operaciones gráficas básicas.

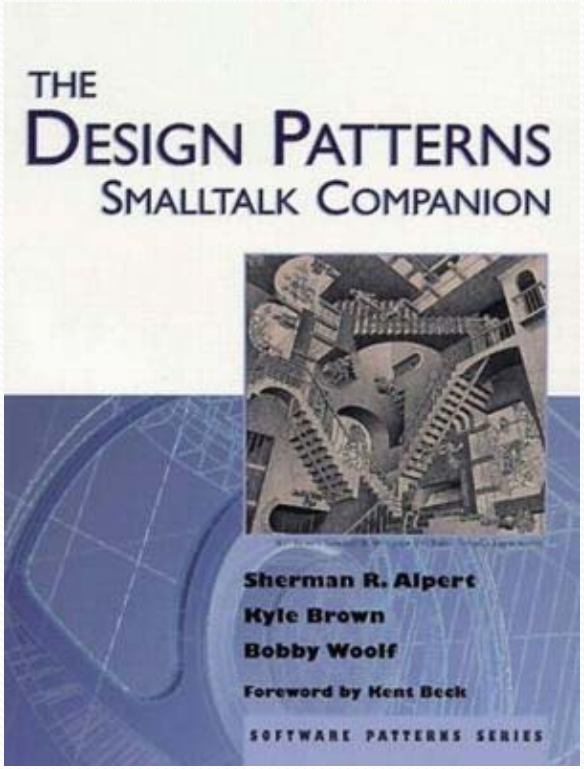
Representación de Objetos

- La representación de un objeto debe contener sólo la información que varía de objeto a objeto.
- La información que sea igual para una clase de objetos se almacena en la representación de dicha clase.

Representación de Objetos

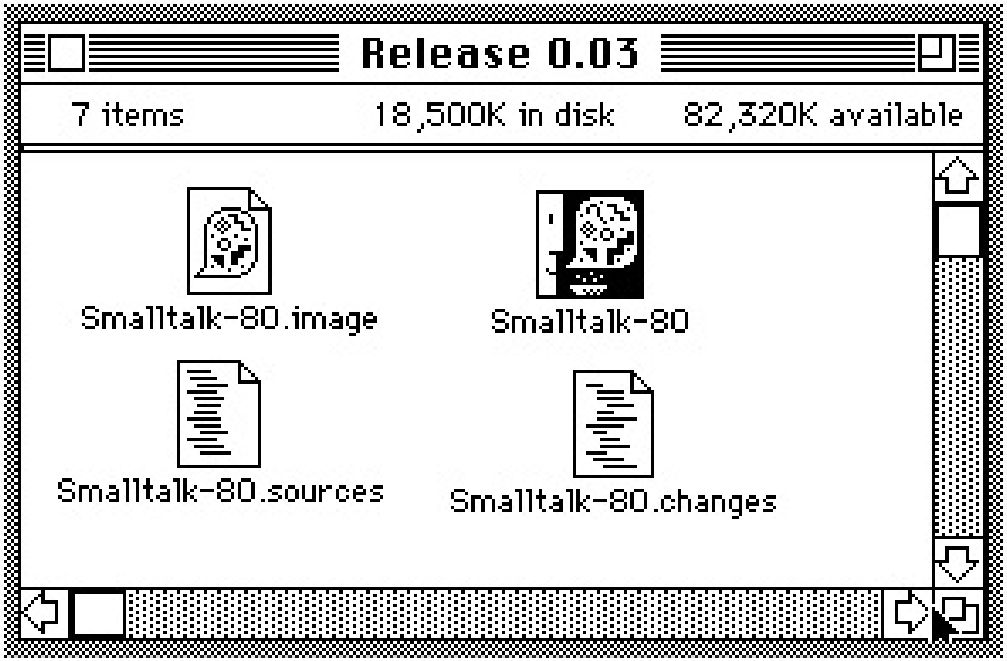
- La información que varía entre las instancias de una clase es simplemente la correspondiente a las variables de instanciación.
- La información almacenada con la clase incluye los métodos de la clase y los métodos de la instancia.

Representación de Objetos



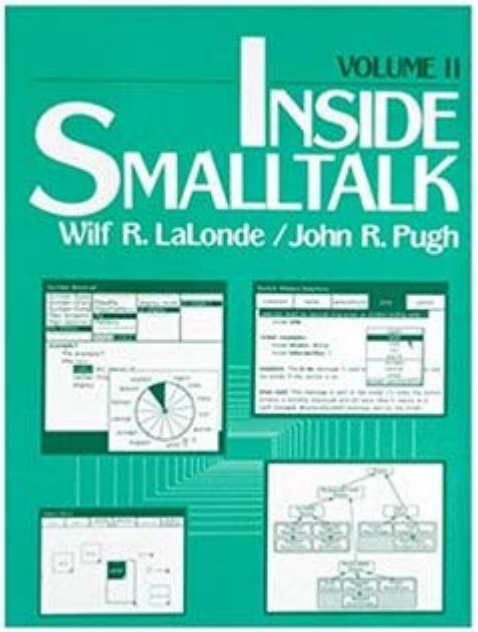
- Puesto que necesitamos saber la clase a la que pertenece cada objeto, la representación de un objeto debe contener alguna indicación de su clase correspondiente.

Representación de Objetos



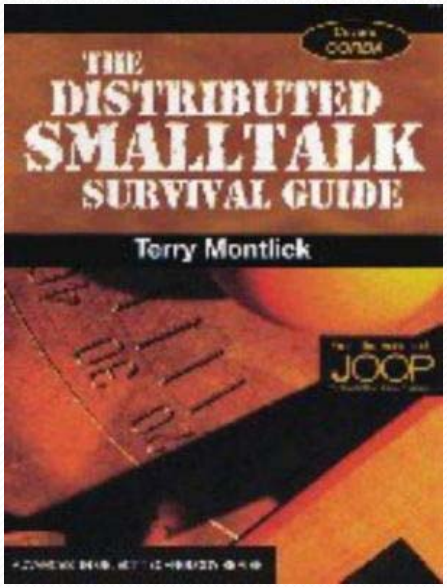
- Hay varias formas de hacer eso: la forma más simple es incluir en la representación del objeto un apuntador a la estructura de datos que representa la clase.

Representación de Objetos



- Además de las variables de instanciación y de la descripción de la clase, cada objeto tiene un campo de longitud requerido por el manejador de almacenamiento para asignar, liberar y mover objetos en memoria.

Representación de Clases



- Puesto que el Smalltalk mantiene la regularidad en todo momento, no debiera sorprendernos que las clases sean también objetos, los cuales son instanciaciones de la metaclase.

Representación de Clases

- Por lo tanto, las clases se representan en la misma forma que los objetos, pero con un apuntador a la metaclase.

- Las variables instanciadas de una clase contienen apuntadores a los objetos que representan la información que es igual para todas las instancias de la clase.

Representación de Clases

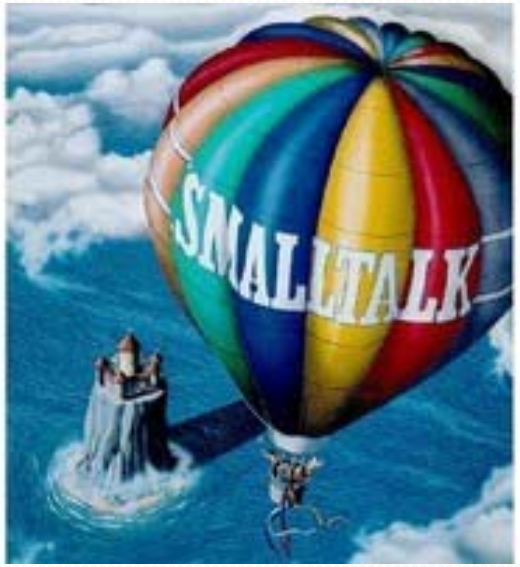
- Esta información incluye lo siguiente:
  - El nombre de la clase
  - La superclase (la cual es "**object**" si no se especifica otra)
  - Los nombres de las variables instanciadas
  - El diccionario de mensajes instanciados

Representación de Clases

- Advierta que en este caso es necesario conocer el número de variables instanciadas para determinar la cantidad de almacenamiento requerida.

- Si no tuviésemos este campo, sería más lento efectuar instanciaciones de objetos.

Representación de Clases



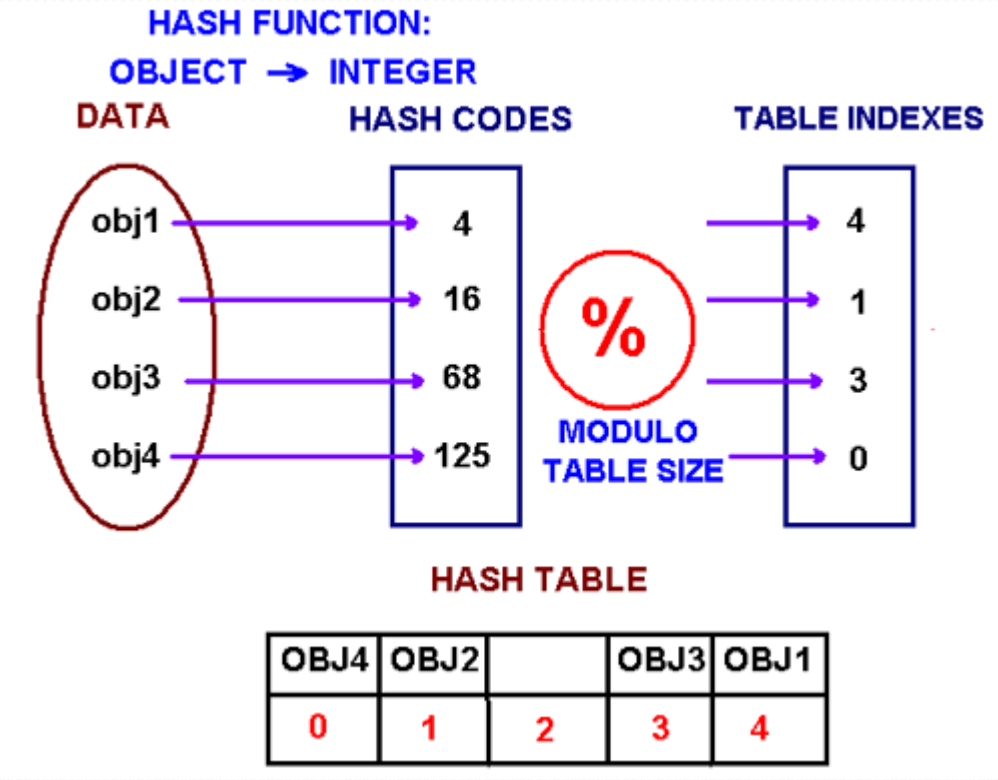
- Puesto que sería demasiado lento que el intérprete tuviese que decodificar el formato fuente de un método cada vez que el método fuese ejecutado, es una buena idea compilar los métodos en alguna forma de pseudo-código que pueda interpretarse rápidamente.

Representación de Clases

- Por otra parte, el formato fuente de los métodos se requiere para editar y desplegar las definiciones de la clase.

- Por lo tanto, los diccionarios de mensajes contienen dos entradas para cada formato del mensaje: una que contiene el formato fuente del método y otra que contiene una forma compilada.

Representación de Clases



- Para encontrar estas entradas en el diccionario de mensajes, se utilizan técnicas de “hashing”.

Representación de los Registros de Activación

- El paso de mensajes en Smalltalk es muy similar a la invocación de procedimientos en los lenguajes estructurados.

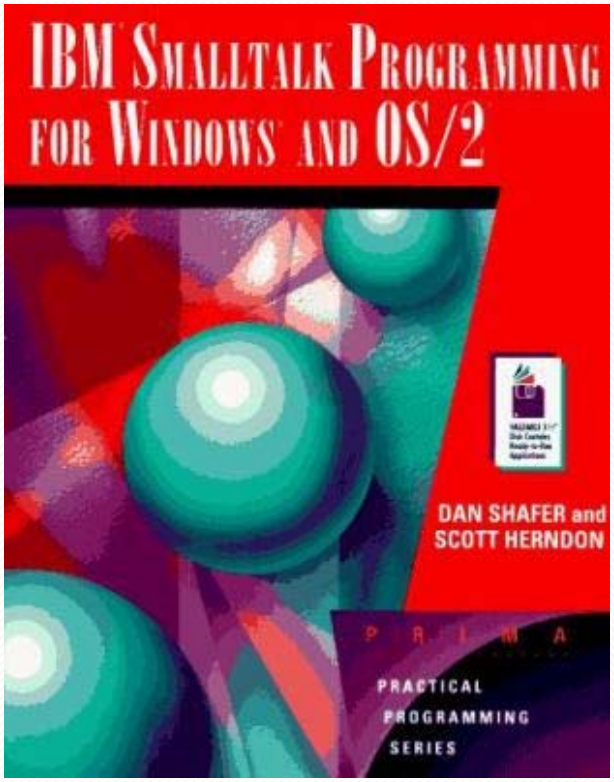
- Por lo tanto, no debiera sorprendernos que se usen técnicas de implementación muy similares, aunque con algunas diferencias importantes.

Representación de los Registros de Activación

- Hemos visto que los registros de activación (RAs) son el vehículo principal para la implementación de procedimientos.

- Puesto que un RA tiene toda la información pertinente a una activación del procedimiento, el proceso de invocar un procedimiento y regresar de él puede entenderse como la manipulación de RAs.

Representación de los Registros de Activación



- Lo mismo aplica a Smalltalk: usaremos registros de activación para conservar toda la información relevante a una activación de un método.

Representación de los Registros de Activación

- La estructura de un RA en Smalltalk es muy similar a la usada en los lenguajes estructurados:

  1) **Parte del ambiente:** El contexto a usarse para la ejecución del método.

Representación de los Registros de Activación

2) **Parte de la instrucción:** La instrucción a ejecutarse cuando este método continúa.

3) **Parte del remitente:** El registro de activación del método que envió el mensaje invocando este método (esto es similar a la liga dinámica).

Representación de los Registros de Activación

- Se usa un sistema de coordenadas en dos dimensiones para identificar instrucciones:
  - *Un apuntador al objeto* identifica el método-objeto que contiene todas las instrucciones de un método.
  - *Un desplazamiento relativo* que identifica la instrucción en particular dentro del par método-objeto.