Lenguajes de Programación

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

Aspectos Sintácticos

- Ada es un lenguaje que busca imponer el **Principio de la Consistencia Sintáctica** en la mayor parte de sus estructuras sintácticas:

- Cosas que se ven similares, deben ser similares; cosas que son diferentes, deben verse diferentes.

Aspectos Sintácticos

- Ada sigue la sintaxis de Pascal, aunque en algunos casos la modifica para hacerla más sistemática.
- Por ejemplo, veamos la sintaxis de los bloques y procedimientos en Ada en el acetato siguiente.

Aspectos Sintácticos

```
declare
  <local declarations>
begin
    <statements>
exceptions
    <exception handlers>
end;
```

```
procedure <name> (<formals>) is
        <local declarations>
        begin
            <statements>
        exceptions
        <exception handlers>
        end;
```

Aspectos Sintácticos

- La diferencia entre los procedimientos y los bloques es que los primeros tienen un nombre y parámetros (y por lo tanto pueden ser invocados desde diferentes contextos con diferentes parámetros), mientras que los bloques no.

- Hemos visto que las funciones y los cuerpos de las tareas siguen un patrón similar.

Aspectos Sintácticos

- Hay otros casos donde Ada ha hecho que las similitudes sintácticas reflejen similitudes semánticas.
- Por ejemplo, la notación que usa Ada para los registros variantes, la cual define varios casos diferentes para un registro, y que se hace lucir como una sentencia de casos.
- Así mismo, la notación para los parámetros independientes de la posición es similar a la que se usa para inicializar arreglos y otras estructuras de datos compuestas.

Aspectos Sintácticos

- Otra diferencia entre Pascal y Ada es en torno al uso del punto y coma. En Pascal, al igual que en Algol, el punto y coma se usa como un *separador*: se coloca entre sentencias. Por tanto, hay una similitud entre los operadores infijos que separan sus operandos:

$$(E_1 + E_2 + E_3)$$

y el punto y coma usado para separar sentencias:

begin S_1 ; S_2 ; S_3 **end**

Aspectos Sintácticos

- Esto crea algunos pequeños problemas de mantenimiento, puesto que si necesitamos insertar una nueva sentencia antes del **end** en una sentencia compuesta como la siguiente:

begin
S_1;
S_2;
S_3
end

debemos recordar agregar un punto y coma a S_3.

Aspectos Sintácticos

- Afortunadamente, tanto Algol como Pascal permiten sentencias vacías, por lo cual podemos escribir:

begin
  S_1;
  S_2;
  S_3;
end

Aunque no es visible, hay una sentencia vacía entre el último punto y coma y el **end**. Muchos programadores en Pascal tendían a terminar todas las sentencias con punto y coma, para mayor regularidad. Esto, además simplifica la edición de los programas.

Aspectos Sintácticos

- Recordemos que una de las contribuciones de Algol fue la sentencia compuesta, que permite que una sentencia contenga otras sentencias.
- Esto condujo a la capacidad de estructurar programas de manera jerárquica y llevó eventualmente a la programación estructurada.
- Sin embargo, la sentencia compuesta tenía una falla.

Aspectos Sintácticos

- Veamos algunos constructores de Pascal:

for i := **... do begin ... end**

if ... then begin ... end else begin ... end

procedure ... begin ... end

function ... begin ... end

case ... of a: **begin ... end; ... end**

while ... do begin ... end

with ... do begin ... end

record ... end

Aspectos Sintácticos

- El problema con esta sintaxis es que todos los constructores terminan con **end**.
- Por tanto, si se omite un **end** por accidente, será difícil localizar cuál es el constructor al que le hace falta.
- Una solución posible sería usar diferentes tipos de paréntesis para cada constructor, pero una más elegante es el uso de una sintaxis completamente acotada por paréntesis.

Aspectos Sintácticos

- Ada tiene pues una sintaxis completamente acotada por paréntesis (*fully bracketed syntax*).

- Esto significa que cada constructor tiene su propio tipo de etiqueta, que hace las veces de un paréntesis.

Aspectos Sintácticos

- Por ejemplo:

loop ... **end loop**;

if ... **end if**;

case ... **end case**;

record ... **end record**;

Aspectos Sintácticos

- Para los constructores que reciben nombre, tales como los subprogramas, paquetes, entradas y tareas, se usa un único tipo de paréntesis, agregando el nombre a la palabra "**end**".

- Veamos algunos ejemplos en el acetato siguiente.

Aspectos Sintácticos

function <name> (<formals>) **is** ... **begin** ... **end** <name>;

procedure <name> (formals>) **is** ... **begin** ... **end** <name>;

package <name> **is** ... **end** <name>;

package body <name> **is** ... **end** <name>;

accept <name> (<formals>) **do** ... **end** <name>;

Aspectos Sintácticos

- Esto permite al compilador realizar un mejor chequeo de errores, puesto que puede asegurarse que cada "**end**" se encuentre ligado a una declaración correcta.
- Un efecto colateral interesante de esta sintaxis es que interactúa con la sentencia "**if**" cuando tenemos varias condiciones "**if-then-else**".
- Veamos un ejemplo en el acetato siguiente.

Aspectos Sintácticos

```
if C1 then
  P1
else
        if C2 then
                P2
        else
                if C3 then
                        P3
                else
                        P4
                end if;
        end if;
end if;
```

Aspectos Sintácticos

- Esto no refleja adecuadamente la intención del programador y viola el **Principio Estructural**.

- Por esta razón, Ada proporciona una sintaxis similar a la de las sentencias de casos para expresar encadenamientos de "**else-if**", como se indica en el acetato siguiente.

Aspectos Sintácticos

if C_1 **then**

P_1

elsif C_2 **then**

P_2

elsif C_3 **then**

P_3

else

P_4

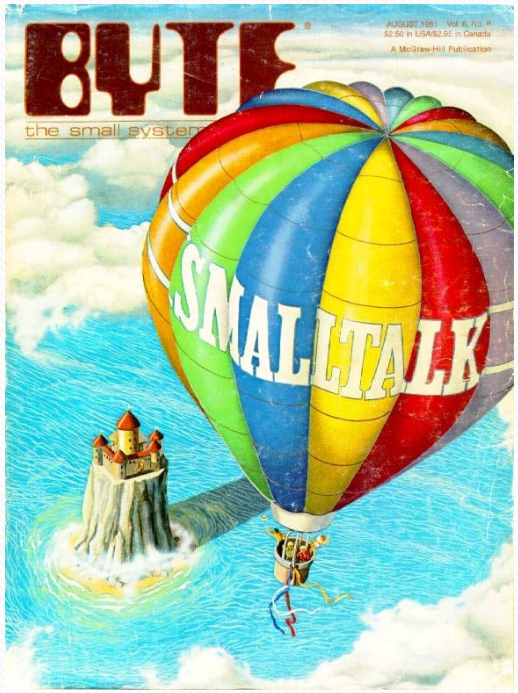
end if;

Aspectos Sintácticos

- Esta idea fue tomada de LISP y conforma mejor tanto el **Principio Estructural** como el **Principio de Consistencia Sintáctica**.

- Sin embargo, su costo es que incrementa un poco la complejidad sintáctica, con lo que se viola ligeramente el **Principio de la Simplicidad**.

Algo de Historia del Smalltalk

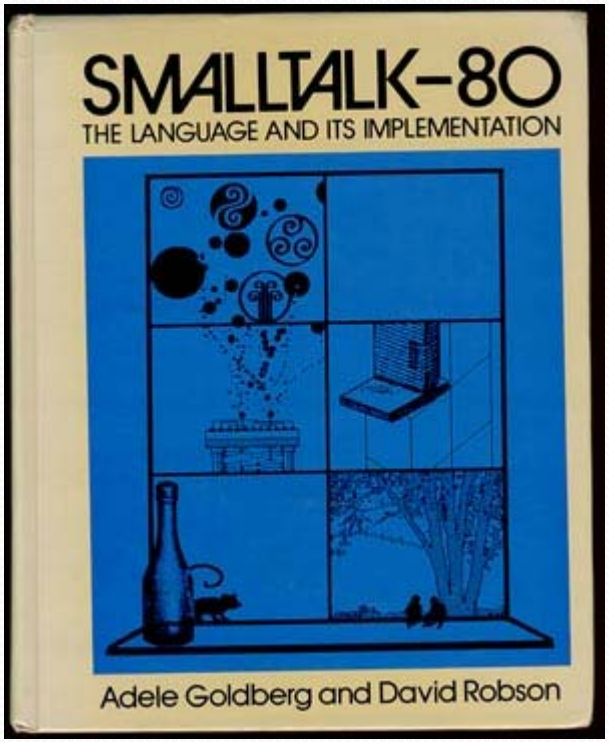


- El concepto de programación orientada a objetos tiene sus raíces en Simula 67, pero se desarrolló completamente en la evolución del lenguaje Smalltalk.

Algo de Historia del Smalltalk

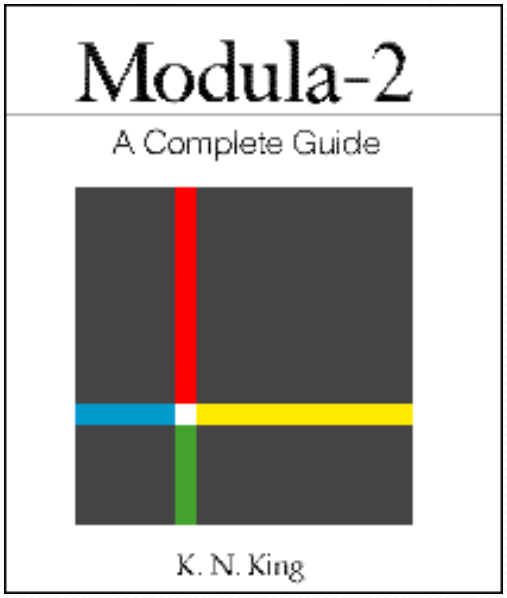
- La programación orientada a los datos se enfoca en el uso de tipos de datos abstractos.
- En este paradigma, la definición estructural y la implementación de los procesos se ocultan de las unidades del programa que las usan.

Algo de Historia del Smalltalk



- Enfocarse en los datos es lo opuesto del enfoque procedural tradicional que se enfoca precisamente en los procesos y su implementación en subprogramas.

Algo de Historia del Smalltalk



- El paradigma de programación orientada a los datos fue popular en los 1980s, y está bien ilustrado en las facilidades de abstracción de datos proporcionadas por Modula-2 y Ada.

Algo de Historia del Smalltalk

- Una restricción fundamental de los tipos de datos abstractos es que, una vez definidos, no pueden ser convenientemente modificados para aplicaciones ligeramente diferentes.
- Tampoco hay manera de colectar las características comunes de una familia de tipos cercanamente relacionados.

Algo de Historia del Smalltalk

- Los lenguajes para programación orientada a objetos extienden la abstracción de datos con el mecanismo de **herencia** para poder proporcionar estas capacidades.

- Ni Ada ni Modula-2 proporcionan mecanismos de herencia.

Algo de Historia del Smalltalk

- La herencia comenzó, de forma limitada, en Simula 67, cuyas clases pueden ser definidas en jerarquías.

- En los lenguajes de tipos estáticos tales como Simula 67 y C++, la herencia es una generalización de la idea de subtipos.

Algo de Historia del Smalltalk

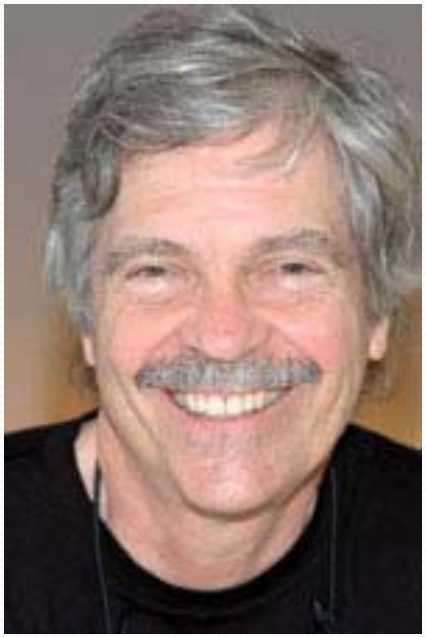
- En Ada, un subtipo del tipo interno INTEGER hereda todas las operaciones de INTEGER.
- La herencia permite que tanto las clases internas como las definidas por el usuario puedan heredar las características de las clases existentes.

Algo de Historia del Smalltalk

- El resultado es que los nuevos programas suelen construirse como variaciones de las unidades existentes, que en este caso son las clases.

- Esto proporciona una técnica efectiva para re-uso de software.

Algo de Historia del Smalltalk

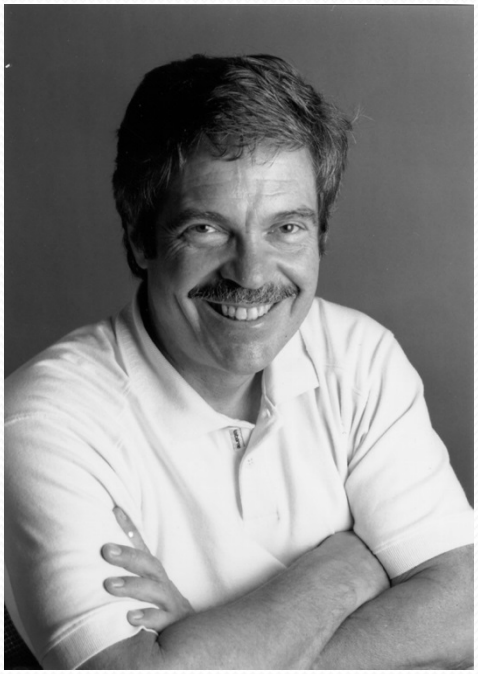


- El principal responsable del desarrollo de Smalltalk fue Alan Kay.

Algo de Historia del Smalltalk

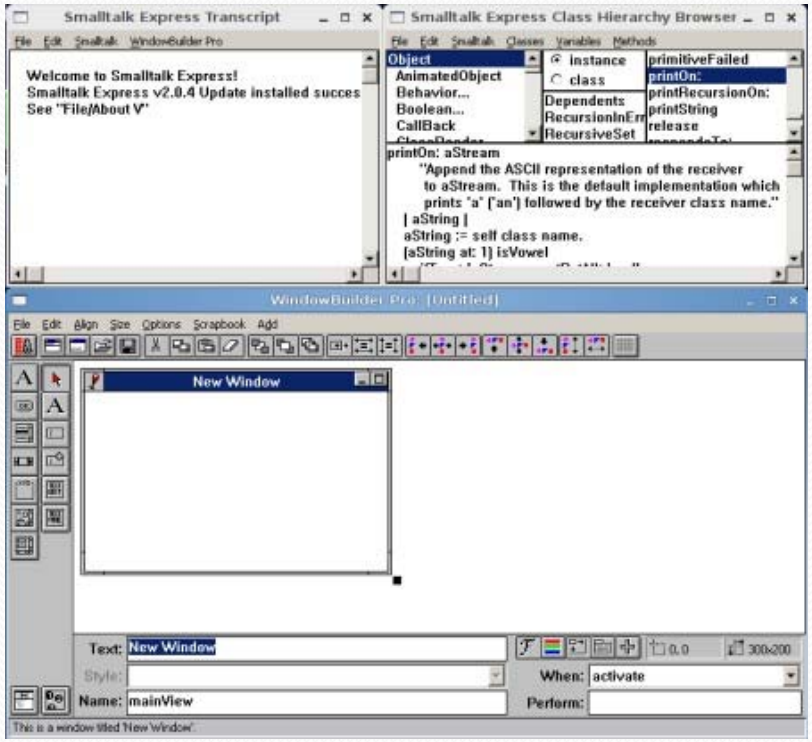
- Cuando Kay era un estudiante de doctorado en la Universidad de Utah, a fines de los 1960s, se convenció de que algún día sería posible poner el poder de lo que en ese entonces era una computadora de $1 millón de dólares, en una notebook.

Algo de Historia del Smalltalk



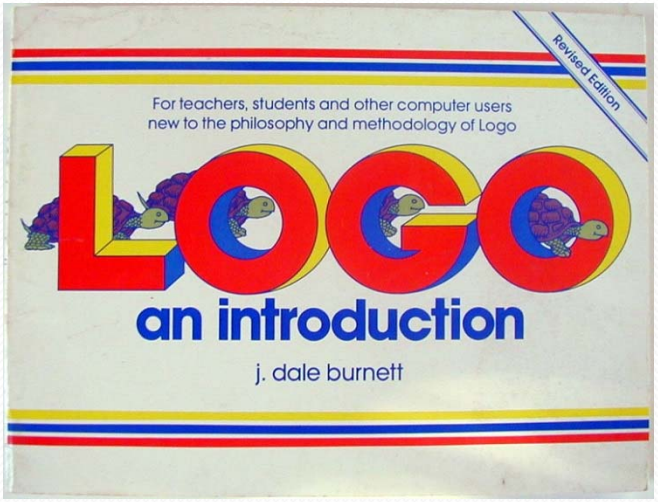
- Kay profetizó que la idea de una computadora en cada hogar se volvería una realidad en unos años más y pensó que los lenguajes de programación de la época resultarían ineficientes debido a que estaban dirigidos a usuarios con una formación más técnica.

Algo de Historia del Smalltalk



- Su idea fue proporcionar un ambiente altamente interactivo y usar gráficos sofisticados para proporcionar una interfaz más amigable con el usuario.

Algo de Historia del Smalltalk

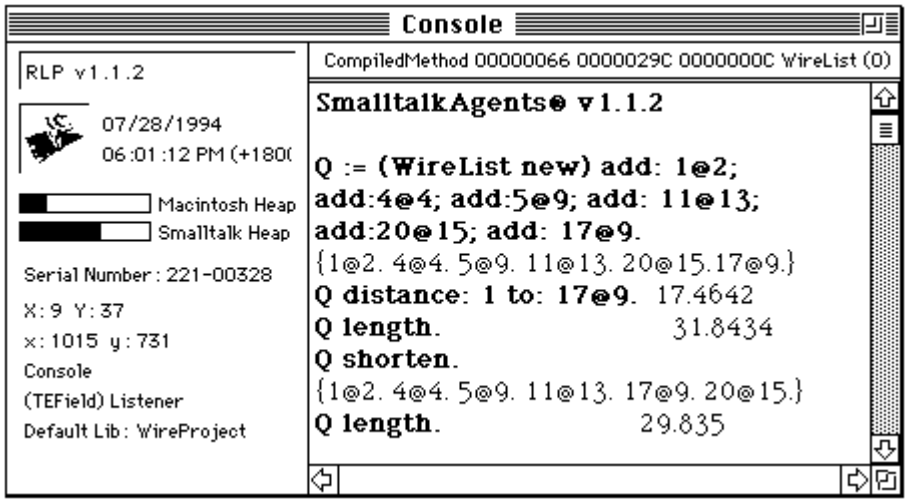


- Kay se vio notablemente influenciado en el diseño de Smalltalk en lenguajes tales como FLEX (un lenguaje que él mismo ayudó a diseñar), LOGO (un lenguaje diseñado por Seymour Papert en el MIT para enseñar a los niños a programar) y Simula 67 (el primer lenguaje orientado a objetos, que estaba basado en Algol 60).

Algo de Historia del Smalltalk

- El lenguaje de programación FLEX todavía estaba muy orientado a los especialistas, por lo que Kay decidió tomar algunas ideas de LOGO para hacerlo más fácil de usar. Así nació Smalltalk.

Algo de Historia del Smalltalk



- El concepto original de Kay era un sistema al que él denominó *Dynabook*, el cual se basaba en el paradigma del escritorio típico, en el cual comúnmente hay varios papeles, algunos de los cuales están parcialmente cubiertos.

Algo de Historia del Smalltalk

- La hoja que se encuentra hasta arriba suele ser el foco de atención, mientras que las demás están temporalmente fuera de foco.
- Por lo tanto, la pantalla del *Dynabook* modelaría precisamente esta escena, usando el concepto de ventanas de pantalla.

Algo de Historia del Smalltalk

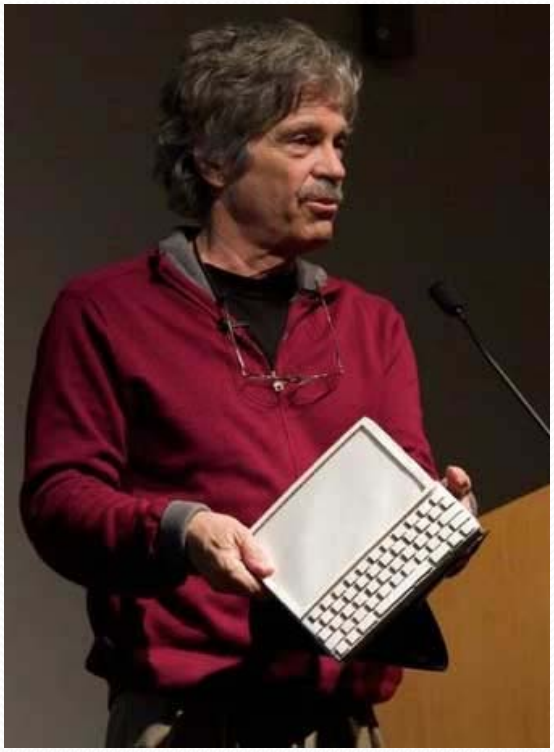
- El usuario interactuaría con este dispositivo a través de un teclado y tocando la pantalla con sus dedos.
- También se pretendía que el *Dynabook* se pudiera conectar a un estéreo, de manera que pudiese usarse para generar música.

Algo de Historia del Smalltalk

- También se podría conectar a las líneas de comunicación, a fin de proporcionar acceso a bancos de datos compartidos de gran tamaño.

- Es decir, que el *Dynabook* estaría conectado en red.

Algo de Historia del Smalltalk



- Después de que el diseño preliminar del *Dynabook* le valió obtener el doctorado en computación a Alan Kay, su meta pasó a ser el ver construida esta máquina.

Algo de Historia del Smalltalk

- Kay se presentó a solicitar trabajo en el *Xerox Palo Alto Research Center* (Xerox PARC).
- Ahí presentó sus ideas sobre el Dynabook, y éstas le valieron ser contratado.
- Con su incorporación, nació el *Learning Research Group* de Xerox.

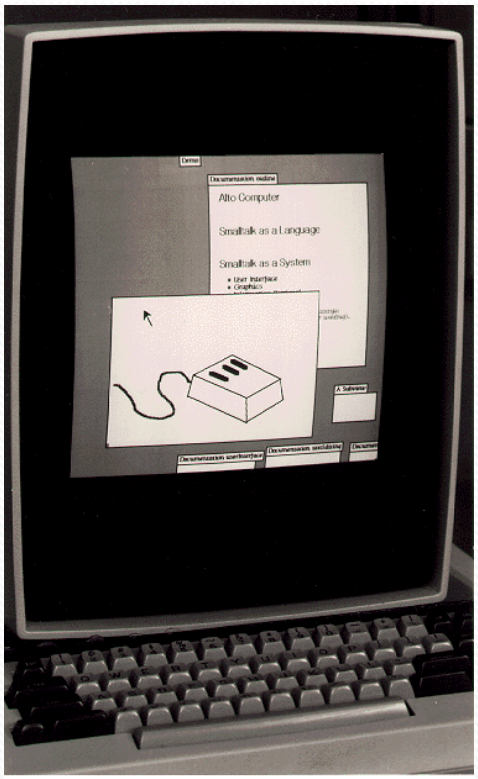
Algo de Historia del Smalltalk

- La primera tarea del grupo fue diseñar un lenguaje que apoyara la idea del paradigma de programación propuesto por Kay.
- Además, el grupo implementaría dicho lenguaje en la mejor computadora personal disponible en aquel entonces.

Algo de Historia del Smalltalk

- Estos esfuerzos produjeron un *Dynabook* "temporal", el cual consistía en hardware de la computadora Alto, de Xerox, y en software del Smalltalk-72.

Algo de Historia del Smalltalk



- Juntos, se convirtieron en una herramienta de investigación que condujo a nuevos desarrollos y a depurar el software y hardware disponibles.

Algo de Historia del Smalltalk

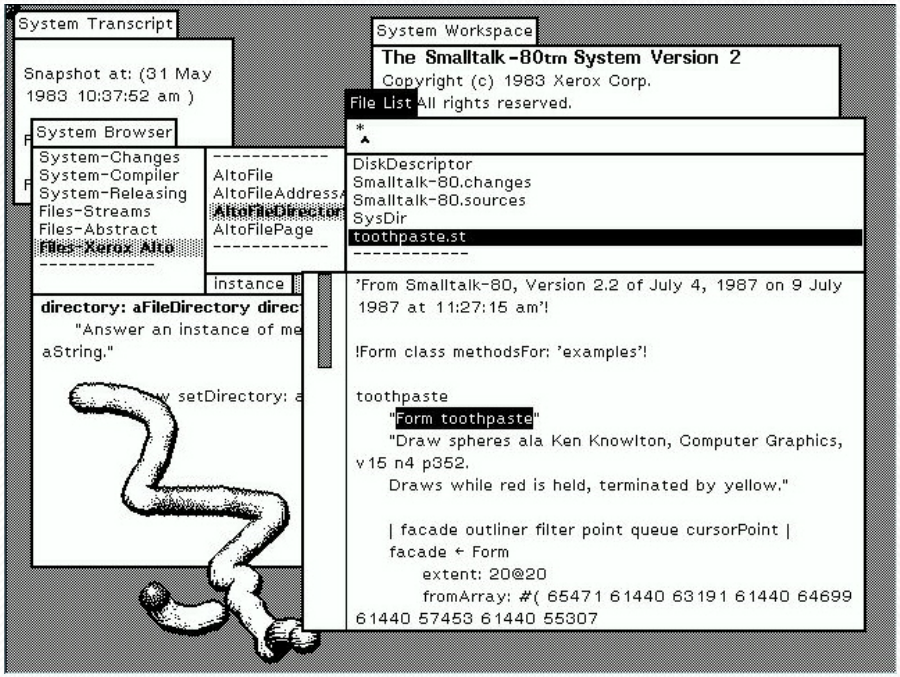
- En 1973, estuvo disponible un *Dynabook* temporal del tamaño de un escritorio.
- Ambos se usaron en diversos experimentos computacionales que involucraron a más de 250 niños (de entre 6 y 15 años de edad) y a 50 adultos.

Algo de Historia del Smalltalk

- A la par de estos experimentos se produjeron nuevos desarrollos que condujeron al Smalltalk-74, Smalltalk-76, Smalltalk-78 y, finalmente, Smalltalk-80.

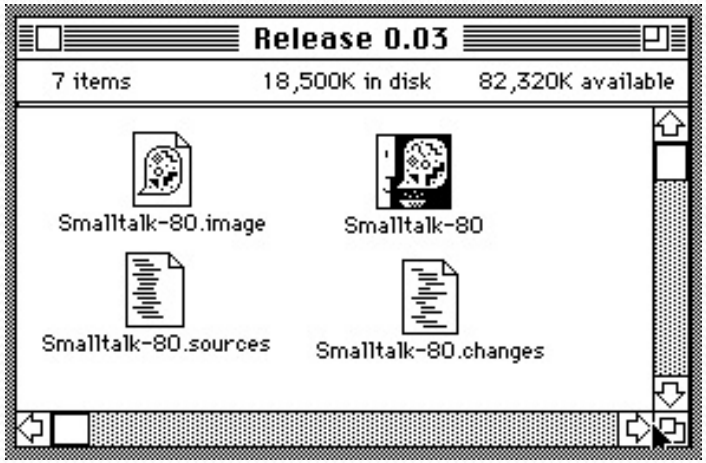
- Esta última versión es la que discutiremos en este curso.

Organización Estructural



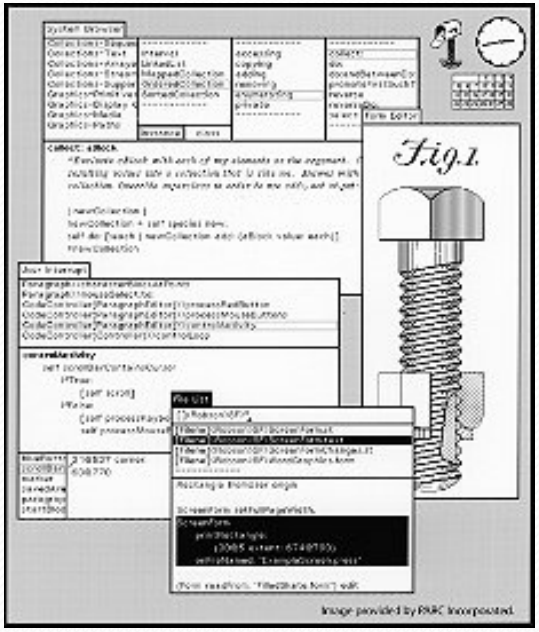
- Una pantalla típica del *Dynabook* muestra una cierta cantidad de ventanas, que son reminiscencia de papeles dispersos sobre un escritorio.

Organización Estructural



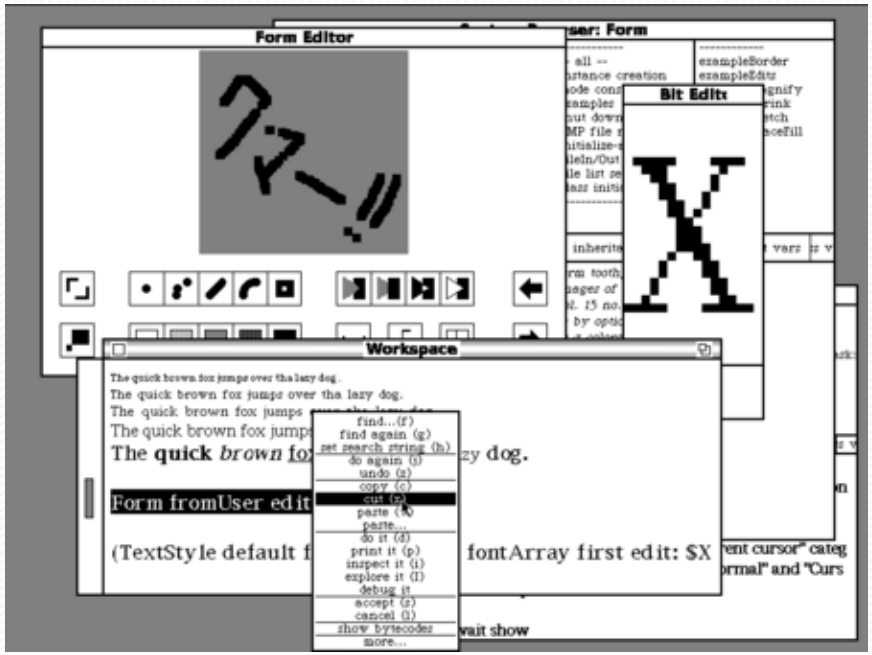
- Estas ventanas contienen muchos tipos de información, incluyendo programas en Smalltalk, salida de programas, correo, documentos en edición, información de depuración, menús, directorios, diagramas, dibujos, e información de estatus (p.ej., la fecha y la hora).

Organización Estructural



- Los usuarios enfocan su atención a una ventana específica apuntando a ella con un cierto dispositivo (normalmente un ratón o el dedo del usuario).

Organización Estructural



- Esto permite a los usuarios trabajar en muchas cosas diferentes a la vez puesto que pueden suspender su actividad en una ventana y continuar otra actividad distinta en otra simplemente moviendo el apuntador de una a otra ventana.

Organización Estructural

- Smalltalk maneja las ventanas de manera que su uso resulte conveniente.
- Por ejemplo, si el usuario “toca” una ventana, entonces se le coloca en la parte superior de otras ventanas que pudieron haber estado ocultándola parcialmente.
- Las ventanas pueden también moverse alrededor de la pantalla usando el ratón.

Organización Estructural

- Smalltalk es un lenguaje altamente interactivo y se implementa como intérprete.
- Aunque mucha de la comunicación con el usuario se realiza a través del ratón, es posible escribir comandos a ejecutarse en una ventana de “diálogo”.

Organización Estructural

- Hay dos formas primarias de definir cosas en Smalltalk. La primera "asocia" un nombre a un objeto. Por ejemplo:

```
x ← 5
y ← x - 2
```

asocia "x" al objeto "5" & "y" al objeto "3".

Organización Estructural

- La otra forma de definir cosas es a través de la "definición de clases".
- Las clases son similares a los paquetes en Ada.
- El usuario solicita la evaluación de una expresión escribiendo una expresión tal como "x*5".

Organización Estructural

- El diálogo sería el siguiente:

```
x*5
25
```

- Smalltalk interpreta esta acción como si se enviara el mensaje "*5" al objeto "x".

Organización Estructural

- Las unidades de programa en Smalltalk son los "objetos".
- Los objetos son estructuras que encapsulan datos locales y un conjunto de operaciones llamadas "métodos", las cuales están disponibles a otros objetos.

Organización Estructural

- Un método especifica la reacción del objeto cuando recibe el mensaje en particular que corresponde a ese método.
- El mundo de Smalltalk está poblado solamente de objetos, desde constantes enteras hasta grandes sistemas de software de enorme complejidad.

Organización Estructural

- Algunas de las ideas fundamentales de Smalltalk son las siguientes:

1) Los objetos tienen un comportamiento.

2) Se puede hacer que los objetos hagan cosas mediante el envío de mensajes.

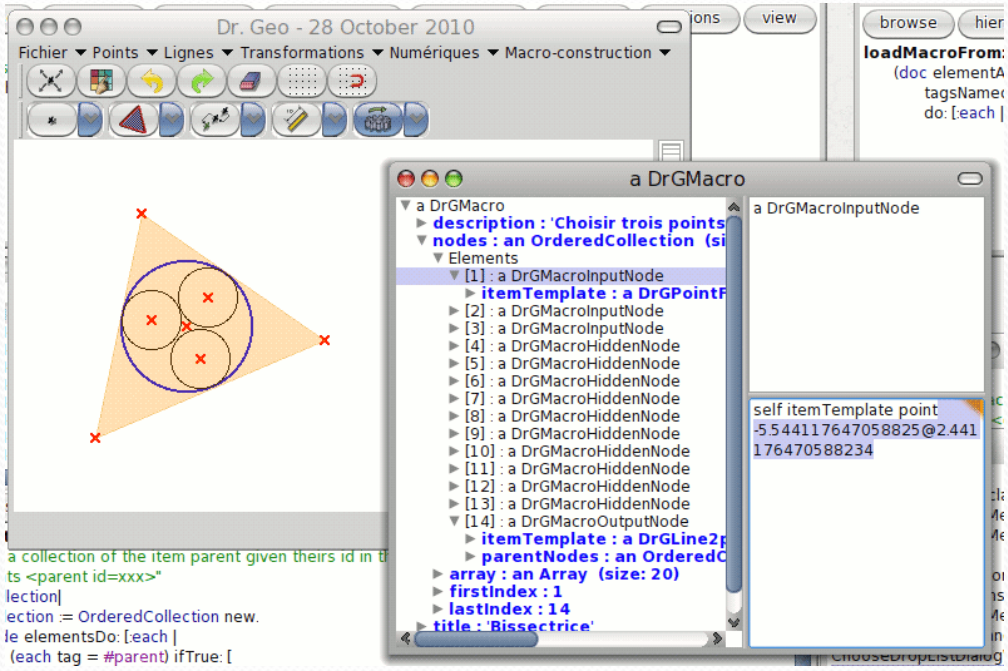
3) Las operaciones repetitivas se pueden simplificar usando estructuras de control.

Organización Estructural

- Todo el cómputo en Smalltalk se realiza mediante la misma técnica uniforme (siguiendo el **Principio de Regularidad**):

- Enviar un mensaje a un objeto para invocar uno de sus métodos.

Organización Estructural



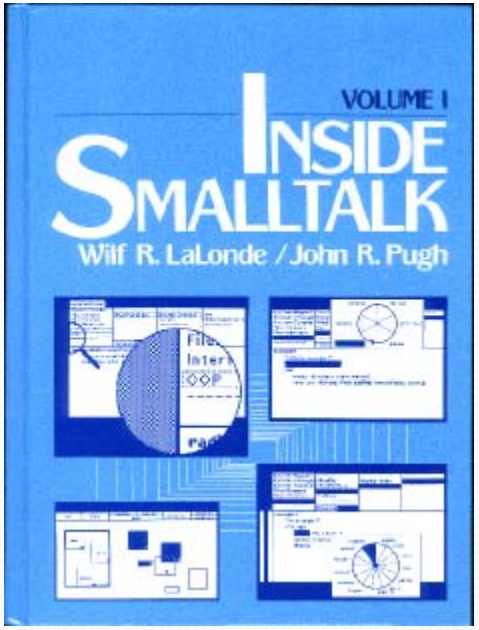
- La réplica a un mensaje es un objeto que retorna la información solicitada, o simplemente notifica al remitente que el procesamiento solicitado ha sido completado.

Organización Estructural

- La diferencia fundamental entre un mensaje y la invocación a un subprograma es la siguiente:

- Se envía un mensaje a un objeto de datos, el cual es luego procesado mediante código asociado con el objeto; una invocación a subprograma usualmente envía los datos a ser procesados a una unidad de código del subprograma.

Organización Estructural



- Desde el punto de vista de la simulación, Smalltalk es una simulación de colecciones de computadoras (objetos) que se comunican entre sí (a través de mensajes).

Organización Estructural

- Cada objeto es una abstracción de una computadora en el sentido de que almacena datos y proporciona capacidad de procesamiento para manipular los datos.

- Adicionalmente, los objetos pueden enviar y recibir mensajes, las cuales son capacidades básicas de una computadora.

Organización Estructural

- En Smalltalk, las abstracciones de objetos son las clases, las cuales son muy similares a las clases de Simula 67.
- Pueden crearse instancias de una clase, las cuales se convierten entonces en los objetos del programa.
- Cada objeto tiene sus propios datos locales y representa una diferente instancia de su clase.

Organización Estructural

- La única diferencia entre dos objetos de la misma clase es el estado de sus variables locales.
- Podemos también definir clases en Smalltalk, y estas nuevas clases pueden ser instanciadas cualquier número de veces, al igual que las clases nativas del lenguaje.

Organización Estructural

- Esto sigue los **Principios de Abstracción y Regularidad**.
- Smalltalk tiene 4 tipos de expresiones: literales, nombres de variables, expresiones de mensajes y expresiones de bloque.

Organización Estructural

- **Literales:** Las literales más comunes son los números, cadenas y palabras clave.

- Los **números** son objetos literales que representan valores numéricos, pero son muy diferentes de las literales numéricas de los lenguajes imperativos tradicionales.

Organización Estructural

- En dichos lenguajes, los números actúan como constantes nombradas, ya que se les asocia con posiciones de memoria que contienen sus valores.

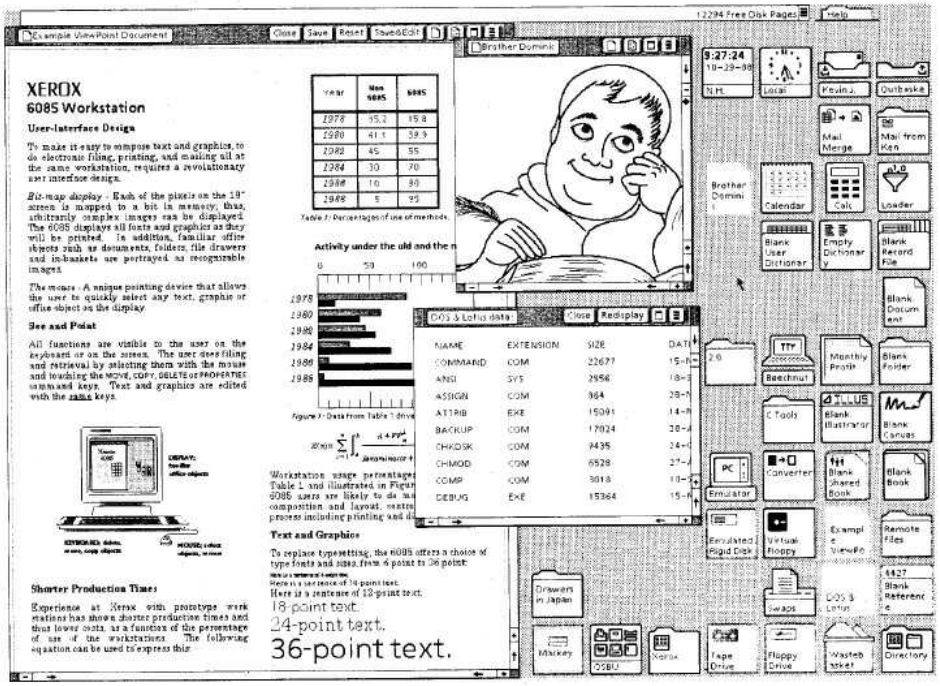
- En Smalltalk, los números son objetos que son caracterizados mediante su protocolo de mensajes y mediante los resultados que producen cuando reciben mensajes.

Organización Estructural

- Sintácticamente, una cadena es una secuencia de caracteres delimitada por apóstrofes.

- Semánticamente, una cadena es un objeto que es capaz de responder a mensajes que accesan a caracteres individuales, reemplazan subcadenas, y efectúan comparaciones con otras cadenas.

Organización Estructural



- Una palabra clave es un identificador, el cual puede ser definido por el usuario, mediante el uso de los dos puntos (:) al final.

Organización Estructural

- **Variables:** Son sintácticamente similares a las de otros lenguajes de programación (una secuencia de letras y/o dígitos que empiezan con una letra).

- Las variables en Smalltalk se dan en dos variedades: *privadas* (locales a un objeto) y *compartidas* (visibles fuera del objeto en el que se "declararon").

Organización Estructural

- Los nombres de las variables privadas deben empezar con minúscula.
- Los nombres de las variables compartidas deben comenzar con mayúscula.

Organización Estructural

- Todas las variables en Smalltalk son apuntadores; sólo pueden referirse a objetos o clases.
- En cierto sentido, no tienen tipo, porque una variable puede apuntar a cualquier objeto.

Organización Estructural

- **<u>Expresiones de Mensaje</u>:** Los mensajes tienen la forma de expresiones.

- Proporcionan el medio de comunicación entre los objetos y son la forma en que se solicitan las operaciones de un objeto.

Organización Estructural

- Las expresiones de mensaje tienen dos partes: una *especificación* del objeto que va a recibir el mensaje, y el *mensaje* en sí.

- El mensaje especifica un selector o método en el objeto receptor y posiblemente uno o más parámetros.

Organización Estructural

- Los parámetros son, al igual que las variables, apuntadores a otros objetos.
- Cuando se evalúa un mensaje, se envía al objeto receptor especificado.

Organización Estructural

- Hay 3 categorías de mensajes: unarios, binarios y palabra clave.
- Los mensajes *unarios* son el tipo más simple, pues no tienen parámetros. Sólo tienen dos partes: el objeto al cual se envían y el método en el objeto receptor.
- El primer símbolo de un mensaje unario especifica un objeto receptor; el último símbolo especifica el método del objeto que será ejecutado.

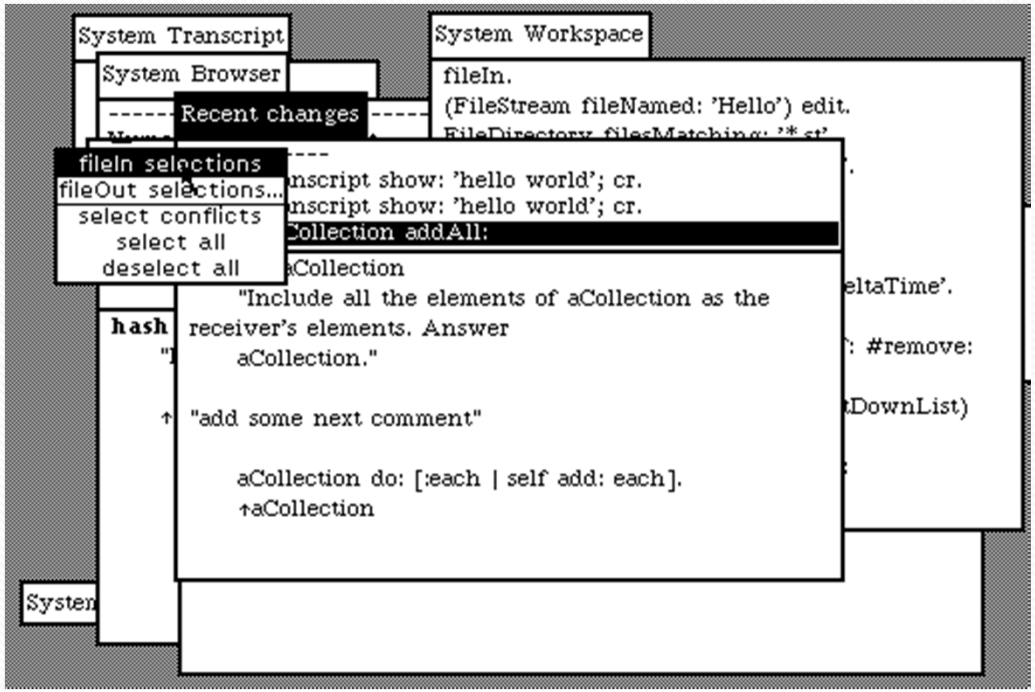
Organización Estructural

- Por ejemplo, el mensaje:

primerAngulo sin

- envía un mensaje sin parámetros al método “sin” del objeto “primerAngulo”.

Organización Estructural



- Puesto que todos los objetos son referenciados mediante apuntadores, "primerAngulo" es realmente un apuntador a un objeto.

Organización Estructural

- Los mensajes *binarios* tienen un solo parámetro: un objeto que es pasado al método especificado del objeto receptor especificado.

- Entre los mensajes binarios más comunes se encuentran las operaciones aritméticas.

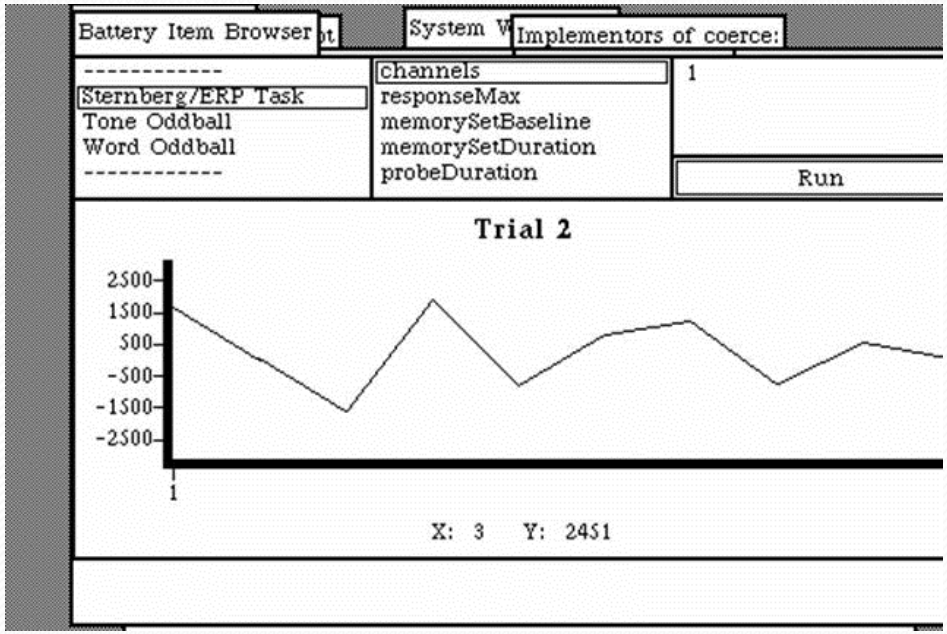
Organización Estructural

- Ejemplos:

2+12

suma/contador

Organización Estructural



- Las *palabras clave* especifican una o más palabras clave que sirven para organizar la correspondencia entre los parámetros enviados en el mensajes y los especificados originalmente en el método.

Organización Estructural

- Es decir, las palabras clave actúan a manera de enlace para seleccionar el método al cual se dirige un mensaje.
- Los métodos que aceptan mensajes de una palabra clave no tienen nombre.
- Tales métodos se identifican mediante las mismas palabras clave.

Organización Estructural

- Considere el siguiente ejemplo:

primerArreglo at: 1 put: 5

- Este mensaje envía los objetos 1 y 5 a un método particular del objeto “primerArreglo”.

Organización Estructural

- Las palabras clave "**at:**" y "**put:**" identifican los parámetros originales del método al cual se enviarán los valores 1 y 5, respectivamente.

- El método al cual se enviará este mensaje incluye las palabras clave del mensaje, las cuales hacen las veces de identificador.

- A este tipo de mensaje se le llama "**selector**".

Organización Estructural

- Las expresiones unarias tienen la precedencia más alta, seguidas de las expresiones binarias y luego de las expresiones con palabras clave.
- Tanto las expresiones unarias como las binarias se asocian de izquierda a derecha.

Organización Estructural

- Esto es muy diferente de las reglas de precedencia usadas en lenguajes tales como Pascal y C.
- Las expresiones, sin embargo, pueden rodearse de paréntesis, a fin de forzar un cierto orden de evaluación de los operadores.

Organización Estructural

- Los mensajes pueden ser enviados en “cascada”.
- Esto significa que pueden enviarse múltiples mensajes al mismo objeto sin tener que duplicar el nombre del objeto receptor.
- Esto se hace separando los grupos de parejas selector-parámetro mediante el uso de punto y coma (;).

Organización Estructural

- Los mensajes se envían secuencialmente, conforme aparecen, de izquierda a derecha. Por ejemplo:

```
ourPen home; up; goto: 500@500; down; home
```

Organización Estructural

- Lo anterior es equivalente a:

```
ourPen home.
ourPen up.
ourPen goto: 500@500.
ourPen down.
ourPen home.
```

Organización Estructural

- Advierta que se usan puntos para separar los mensajes que se envían a métodos diferentes y los cuales aparecen en líneas separadas.

- Esto es similar al uso del punto y coma en Pascal (utilizado para separar sentencias).

Organización Estructural

- **<u>Expresiones de Bloque</u>:** Un bloque es un objeto literal sin nombre que contiene una secuencia de expresiones.
- Los bloques son instancias de la clase **Block**.
- Puede enviarse un mensaje a un bloque colocándolo inmediatamente después del bloque.

Organización Estructural

- Uno de los aspectos más inusuales de Smalltalk es que sus estructuras de control no son proporcionadas por sentencias del lenguaje.
- En vez de eso, se forman con el paradigma fundamental de la orientación a objetos: el paso de mensajes.

Organización Estructural

- Los bloques proporcionan una manera de colectar expresiones a fin de formar grupos.
- Estos grupos pueden luego ser usados para construir estructuras de control.
- Un bloque se especifica mediante el uso de corchetes, separando sus componentes mediante puntos.

Organización Estructural

- Ejemplo:

```
[index ← index + 1. sum ← sum + index]
```

- Las expresiones en un bloque son acciones diferidas, ya que no se ejecutan cuando se les encuentra.

Organización Estructural

- Más bien, los bloques son ejecutados sólo cuando a éstos se les envía el mensaje unario "**value**".

- Por ejemplo:

```
[sum ← sum + index] value
```

Organización Estructural

- Lo anterior envía el mensaje "**value**" al bloque, lo que provoca su ejecución.
- Cuando se completa la ejecución de un bloque, se regresa el valor de la última expresión en el bloque.

Organización Estructural

- Los bloques siempre se ejecutan en el contexto de su definición (asociación dinámica), aun cuando se les envíe como parámetros a un objeto diferente.
- Por lo tanto, están relacionados semánticamente con el mecanismo de paso por nombre de Algol-60.

Organización Estructural

- Los bloques pueden verse como declaraciones de procedimientos que pueden aparecer en cualquier parte.
- Al igual que los procedimientos, los bloques pueden tener parámetros. Dichos parámetros se especifican en una sección al inicio del bloque.
- A esta sección se le separa mediante el uso de una barra vertical (|).

Organización Estructural

- Las especificaciones de parámetros requieren de la colocación de dos puntos (:) al extremo izquierdo de cada parámetro.

- Dado que no hay tipos declarados, estas especificaciones incluyen sólo los nombres de los parámetros del bloque, los cuales se listan sin ningún tipo de separador.

Organización Estructural

- Ejemplo:

```
[:x :y | sum ← x + 10. total ← sum * y]
```

- Los bloques proporcionan un medio para colectar expresiones, de forma que son una manera natural de formar estructuras de control en Smalltalk.

Iteración

- Los bloques pueden contener expresiones relacionales, en cuyo caso regresan uno de los objetos Booleanos predefinidos: **true** o **false**.
- Tales bloques son algunas veces llamados “condicionales”.
- Los dos objetos “true” y “false” tienen métodos que proporcionan algunas de las facilidades para construir estructuras de control.

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.